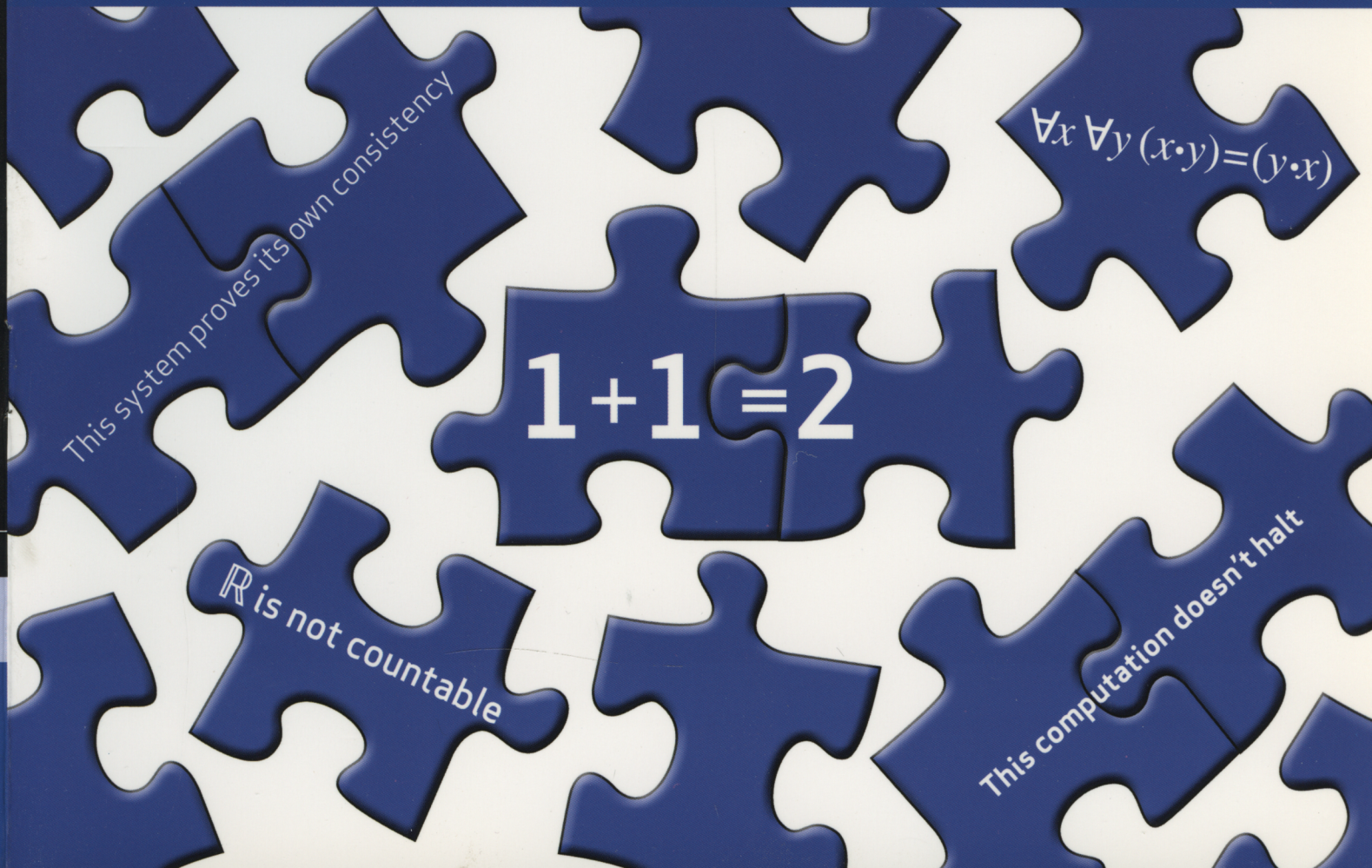
The Open University

M381 Number Theory and Mathematical Logic

Mathematical Logic **Unit 2**

Primitive Recursive Functions



The Open University

M381 Number Theory and
Mathematical Logic

Mathematical Logic **Unit 2**

Primitive Recursive Functions

Prepared by the Course Team

The M381 Mathematical Logic Course Team

The Mathematical Logic half of the course was produced by the following team:

Roberta Cheriyan	*Course Manager*
Derek Goldrei	*Course Team Chair* and *Academic Editor*
Jeremy Gray	*History Consultant*
Mary Jones	*Critical Reader*
Roger Lowry	*Publishing Editor*
Alan Pears	*Author*
Alan Slomson	*Author*
Frances Williams	*Critical Reader*

with valuable assistance from:

The Maths Production Unit, Learning & Teaching Solutions:
Becky Browne, Jim Campbell, Nicky Kempton, Bill Norman, Sharon Powell, Katie Sayce, Penny Tee

Alison Cadle	*TEX Consultant*
Michael Goldrei	*Cover Design Consultant*
Vicki McCulloch	*Cover Designer*

The external assessor was:

Jeff Paris	*Professor of Pure Mathematics, University of Manchester*

The Course Team would like to acknowledge their reliance on the previous work of Alan Slomson and of Alex Wilkie, Professor of Mathematical Logic, University of Oxford.

This publication forms part of an Open University course. Details of this and other Open University courses can be obtained from the Student Registration and Enquiry Service, The Open University, PO Box 197, Milton Keynes, MK7 6BJ, United Kingdom: tel. +44 (0)870 300 6090, e-mail general-enquiries@open.ac.uk

Alternatively, you may visit the Open University website at http://www.open.ac.uk where you can learn more about the wide range of courses and packs offered at all levels by The Open University.

To purchase a selection of Open University course materials, visit http://www.ouw.co.uk, or contact Open University Worldwide, Michael Young Building, Walton Hall, Milton Keynes, MK7 6AA, United Kingdom, for a brochure: tel. +44 (0)1908 858793, fax +44 (0)1908 858787, e-mail ouw-customer-services@open.ac.uk

The Open University, Walton Hall, Milton Keynes, MK7 6AA.

First published 2004. Reprinted as new edition 2007, with corrections.

Copyright © 2004 The Open University

All rights reserved; no part of this publication may be reproduced, stored in a retrieval system, transmitted or utilised in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without written permission from the publisher or a licence from the Copyright Licensing Agency Ltd. Details of such licences (for reprographic reproduction) may be obtained from the Copyright Licensing Agency Ltd, Saffron House, 6–10 Kirby Street, London EC1N 8TS; website http://www.cla.co.uk.

Open University course materials may also be made available in electronic formats for use by students of the University. All rights, including copyright and related rights and database rights, in electronic course materials and their contents are owned by or licensed to The Open University, or otherwise used by The Open University as permitted by applicable law.

In using electronic course materials and their contents you agree that your use will be solely for the purposes of following an Open University course of study or otherwise as licensed by The Open University or its assigns.

Except as permitted above you undertake not to copy, store in any medium (including electronic storage or use in a website), distribute, transmit or re-transmit, broadcast, modify or show in public such electronic materials in whole or in part without the prior written consent of The Open University or in accordance with the Copyright, Designs and Patents Act 1988.

Edited, designed and typeset by The Open University, using the Open University TEX System.

Printed in the United Kingdom by Charlesworth Press, Wakefield.

ISBN 978 0 7492 2265 9

3.1

CONTENTS

Introduction 4

1 Primitive Recursive Functions 5

1.1 Definitions and examples 6

1.2 Characteristic functions 11

1.3 Definition by cases 17

2 Infinite Sets 22

2.1 Cantor's theory of countably infinite sets 22

2.2 Coding URM programs 27

2.3 Uncountably infinite sets 30

3 Primitive Recursive Coding 32

3.1 Enumerating the primes 33

3.2 Coding sequences 40

3.3 Codes of URM programs 43

Summary 46

Objectives 47

Additional Exercises 48

Solutions to the Problems 51

Solutions to Additional Exercises 60

Index 67

INTRODUCTION

In *Unit 1* we described how we study the notion of an algorithm in terms of a theoretical computing machine called an *Unlimited Register Machine* (URM). Recall that we defined a function $f : \mathbb{N}^k \longrightarrow \mathbb{N}$ to be *URM-computable* if there is a URM program which computes the values of f. We began our study of URM-computable functions by giving URM programs to show that certain functions are URM-computable. However, we soon turned to a different approach, that of proving general results about URM-computable functions. This is the approach that we shall continue to pursue in this unit.

In *Unit 1* we proved three key theorems which provide a starting point for this approach. The first is Theorem 2.1, which tells us that certain very simple functions are URM-computable.

Theorem 2.1 of Unit 1

The following functions are URM-computable.

(a) The zero function

$$\begin{aligned} \text{zero} &: \mathbb{N} \longrightarrow \mathbb{N} \\ n &\longmapsto 0 \end{aligned}$$

(b) The successor function

$$\begin{aligned} \text{succ} &: \mathbb{N} \longrightarrow \mathbb{N} \\ n &\longmapsto n+1 \end{aligned}$$

(c) The projection functions

$$\begin{aligned} U_m^k &: \mathbb{N}^k \longrightarrow \mathbb{N} \\ (n_1, n_2, \ldots, n_k) &\longmapsto n_m \end{aligned}$$

for all positive integers m, k with $m \leq k$.

The other two key theorems relate to two processes which generate new functions. The first of these processes is *substitution*. If $f : \mathbb{N}^t \longrightarrow \mathbb{N}$, $g_1 : \mathbb{N}^k \longrightarrow \mathbb{N}$, $g_2 : \mathbb{N}^k \longrightarrow \mathbb{N}$, ..., $g_t : \mathbb{N}^k \longrightarrow \mathbb{N}$ are functions, then the function $h : \mathbb{N}^k \longrightarrow \mathbb{N}$ defined by

$$h(n_1, n_2, \ldots, n_k) = f(g_1(n_1, n_2, \ldots, n_k), g_2(n_1, n_2, \ldots, n_k), \ldots, g_t(n_1, n_2, \ldots, n_k))$$

is said to be obtained from f, g_1, g_2, ..., g_t by *substitution*. The key theorem related to this process tells us that the class of URM-computable functions is closed under substitution.

See Definition 3.1 of *Unit 1* for substitution.

Theorem 3.2 of Unit 1

If the functions $f : \mathbb{N}^t \longrightarrow \mathbb{N}$, $g_1 : \mathbb{N}^k \longrightarrow \mathbb{N}$, $g_2 : \mathbb{N}^k \longrightarrow \mathbb{N}$, ..., $g_t : \mathbb{N}^k \longrightarrow \mathbb{N}$ are all URM-computable and $h : \mathbb{N}^k \longrightarrow \mathbb{N}$ is defined from them by substitution, then h is also URM-computable.

The second process is *primitive recursion.* If $f : \mathbb{N}^k \longrightarrow \mathbb{N}$ and $g : \mathbb{N}^{k+2} \longrightarrow \mathbb{N}$ are functions, then the function $h : \mathbb{N}^{k+1} \longrightarrow \mathbb{N}$ is said to be obtained from f and g by *primitive recursion* if

$$h(n_1, n_2, \ldots, n_k, 0) = f(n_1, n_2, \ldots, n_k),$$

$$h(n_1, n_2, \ldots, n_k, n+1) = g(n_1, n_2, \ldots, n_k, n, h(n_1, n_2, \ldots, n_k, n)).$$

The key theorem related to this process tells us that the class of URM-computable functions is closed under primitive recursion.

> ***Theorem 3.3 of Unit 1***
>
> Let $f : \mathbb{N}^k \longrightarrow \mathbb{N}$ and $g : \mathbb{N}^{k+2} \longrightarrow \mathbb{N}$ be URM-computable functions. Then the function $h : \mathbb{N}^{k+1} \longrightarrow \mathbb{N}$ obtained from f and g by primitive recursion is also URM-computable.

In this unit we shall begin by exploring the power of these theorems. We shall see in Section 1 that they enable us to prove that a whole host of functions is URM-computable. It will begin to look as though any URM-computable function can be obtained from the functions listed in Theorem 2.1 of *Unit 1* using substitutions and primitive recursions. However, this first impression turns out to be wrong, and we explain in Section 2 why the processes of substitution and primitive recursion are not powerful enough to generate all the URM-computable functions. In order to provide this explanation, we shall need to study some of the theory of infinite sets. The missing link — that is, the additional process that we need in order to obtain all the URM-computable functions — is discussed in *Unit 3.* To lead into that discussion, in Section 3 we introduce the concept of *bounded minimization* and prove a number of theorems based on this concept.

This unit is rather longer than the other units in the Mathematical Logic part of the course. You may also find some of the material rather difficult. You should, however, try not to spend much longer studying this unit than others in this part of the course. All the material in this unit is important, but the most important material is in Section 1 and Subsection 3.1, so make sure that you study those parts of the unit carefully. As you progress through Sections 1 and 3, you will notice that the level of formality within arguments decreases; as you study these sections, you should bear in mind that one of the aims of this unit is that you should develop a facility with the less formal style of argument.

See Definition 3.3 of *Unit 1* for primitive recursion. Definition 3.2 of *Unit 1* concerned a function of one variable defined by primitive recursion, which can be thought of as a special case of the general definition with $k = 0$ and the function f replaced by a constant a. We shall henceforward assume that this special case is covered by the general definition.

Theorem 3.4 of *Unit 1* dealt with the case of a function of one variable, which can be thought of as a special case of Theorem 3.3 with $k = 0$ and the function f replaced by a constant a. We shall henceforward assume that this special case is covered by the general theorem.

1 PRIMITIVE RECURSIVE FUNCTIONS

This section introduces the idea of a *primitive recursive function.* Subsection 1.1 provides a definition and some simple examples of this concept. Subsection 1.2 introduces the *characteristic function,* a function which describes a set or a relation and which will prove very useful later in this unit and in *Unit 3.* In particular, in Subsection 1.3, we see how characteristic functions can be used to show that more complicated functions, defined by cases, are primitive recursive.

1.1 Definitions and examples

We are going to look at functions that we can prove to be URM-computable by using the theorems listed in the Introduction. These are the functions that can be obtained from the basic functions given in Theorem 2.1 of *Unit 1* using the operations of substitution and primitive recursion. The functions obtainable in this way form an important class of functions, called *primitive recursive functions*, as primitive recursion is the more significant of the two processes used to obtain them. Thus we begin with the following definitions.

Definition 1.1 Basic primitive recursive functions

The *basic primitive recursive functions* are

(a) The zero function

$$\text{zero}: \mathbb{N} \longrightarrow \mathbb{N}$$
$$n \longmapsto 0$$

(b) The successor function

$$\text{succ}: \mathbb{N} \longrightarrow \mathbb{N}$$
$$n \longmapsto n+1$$

(c) The projection functions

$$U_m^k : \mathbb{N}^k \longrightarrow \mathbb{N}$$
$$(n_1, n_2, \ldots, n_k) \longmapsto n_m$$

for all positive integers m, k with $m \leq k$.

Definition 1.2 Primitive recursive function

A function is said to be *primitive recursive* if it can be obtained from basic primitive recursive functions using the operations of substitution and primitive recursion a finite number of times.

It is an immediate consequence of Definition 1.2 that if a function f can be obtained from functions already known to be primitive recursive by substitution or primitive recursion then f is primitive recursive. We shall often make use of this fact.

The following theorem results immediately from these definitions and Theorems 2.1, 3.2 and 3.3 of *Unit 1*, stated in the Introduction to this unit.

Theorem 1.1

Every primitive recursive function is URM-computable.

Note that it does *not* follow that every URM-computable function is primitive recursive. We discuss this point in the next unit.

We now exploit Theorem 1.1 by giving a catalogue of functions which are primitive recursive and hence URM-computable. Some of the functions in our catalogue are standard functions; others are there because we need to make use of them later.

Example 1.1

The constant function

$$C_1 : \mathbb{N} \longrightarrow \mathbb{N}$$
$$n \longmapsto 1$$

is primitive recursive.

Since $\text{succ}(0) = 1$, we see that

$$C_1(n) = \text{succ}(\text{zero}(n)).$$

Thus C_1 is obtained by substitution from the basic primitive recursive functions succ and zero and so is primitive recursive. ♦

Problem 1.1

Use Mathematical Induction to prove that, for each natural number a, the constant function

$$C_a : \mathbb{N} \longrightarrow \mathbb{N}$$
$$n \longmapsto a$$

is primitive recursive.

The result of Problem 1.1 can be generalized to show that, for each natural number a, the *constant function*

$$C_a^k : \mathbb{N}^k \longrightarrow \mathbb{N}$$
$$(n_1, n_2, \ldots, n_k) \longmapsto a$$

is primitive recursive.

You are asked to prove this in Additional Exercise 1 for this section.

Example 1.2 Addition

We have already shown directly, by giving an appropriate URM program in *Unit 1*, that the function $\text{add} : (n, m) \longmapsto n + m$ is URM-computable. Now we shall show that it is primitive recursive by showing that it can be defined by primitive recursion from functions we already know to be primitive recursive.

We observe that

$$\text{add}(n, 0) = n + 0 = n$$

and that

$$\text{add}(n, m + 1) = n + (m + 1) = (n + m) + 1 = \text{succ}(\text{add}(n, m)).$$

To show that add is defined by primitive recursion, we need to produce primitive recursive functions $f : \mathbb{N} \longrightarrow \mathbb{N}$ and $g : \mathbb{N}^3 \longrightarrow \mathbb{N}$ such that

$$\text{add}(n, 0) = f(n),$$
$$\text{add}(n, m + 1) = g(n, m, \text{add}(n, m)).$$

Since $\text{add}(n, 0) = n$, we see that f must be the function given by $f(n) = n$: that is, f must be the basic primitive recursive function U_1^1 (the identity function).

Since $\text{add}(n, m + 1) = \text{succ}(\text{add}(n, m))$, we see that g must be a function which maps the triple $(n, m, \text{add}(n, m))$ to $\text{succ}(\text{add}(n, m))$. A suitable function is thus $g : \mathbb{N}^3 \longrightarrow \mathbb{N}$ given by

$$g(n_1, n_2, n_3) = \text{succ}(n_3),$$

for then

$$g(n, m, \text{add}(n, m)) = \text{succ}(\text{add}(n, m)).$$

Thus to show that add is primitive recursive, it remains to show that g is primitive recursive. To do this we can use the projection function U_3^3 to write

$$g(n_1, n_2, n_3) = \text{succ}(U_3^3(n_1, n_2, n_3)).$$

This shows that g is obtained by substitution from the basic primitive recursive functions succ and U_3^3 and so is primitive recursive.

Thus add is obtained by primitive recursion from the primitive recursive functions U_1^1 and g, and so is a primitive recursive function. ♦

Notice how the projection function U_3^3 is used to obtain a formula for g that satisfies the definition of substitution, which requires that the function(s) substituted into the right-hand-side function (succ here) are of the same number of variables as the left-hand-side function (g here).

Example 1.3 Multiplication

We shall show that the function $\text{mult} : (n, m) \longmapsto n \times m$ is primitive recursive.

We observe that

$$\text{mult}(n, 0) = n \times 0 = 0$$

and that

$$\text{mult}(n, m+1) = n \times (m+1) = (n \times m) + n = \text{add}(\text{mult}(n, m), n).$$

From these equations we shall see that mult is obtained by primitive recursion from known primitive recursive functions.

First we note that

$$\text{mult}(n, 0) = 0 = C_0(n),$$

where C_0 is primitive recursive (by Problem 1.1).

We could equally well have written the zero function $\text{zero}(n)$ instead of the constant function $C_0(n)$.

Next we require a function $g : \mathbb{N}^3 \longrightarrow \mathbb{N}$ such that

$$\text{mult}(n, m+1) = g(n, m, \text{mult}(n, m)).$$

Since $\text{mult}(n, m+1) = \text{add}(\text{mult}(n, m), n)$ we see that a suitable function g is given by

$$g(n_1, n_2, n_3) = \text{add}(n_3, n_1).$$

Using projection functions we can write this as

$$g(n_1, n_2, n_3) = \text{add}\big(U_3^3(n_1, n_2, n_3), U_1^3(n_1, n_2, n_3)\big).$$

Notice again how projection functions are used to provide a formula for g that satisfies the definition of substitution.

This shows that g is obtained by substitution from add, shown to be primitive recursive in Example 1.2, and the basic primitive recursive functions U_3^3 and U_1^3. So g is a primitive recursive function.

Thus mult is obtained by primitive recursion from the primitive recursive functions C_0 and g. Therefore mult is a primitive recursive function. ♦

One of the techniques employed in Examples 1.2 and 1.3 was to use projection functions to obtain formulas that satisfied the definition of substitution. Projection functions can commonly be used in this fashion, to provide functions of the appropriate number of variables, when demonstrating that a function is obtained by substitution or primitive recursion from known primitive recursive functions. In a sense, projection functions are used to pick out a particular variable or to reorder variables.

Problem 1.2

Show that, for fixed natural numbers a and b, the linear function given by $n \longmapsto an + b$ is primitive recursive.

The use of constants, such as a and b in Problem 1.2, is very common in defining functions, so that we shall benefit from generalizing the way that the constant functions and projection functions were used in our solution.

Problem 1.3

Suppose that the function $f : \mathbb{N}^3 \longrightarrow \mathbb{N}$ is primitive recursive and that a is a fixed natural number. Show that each of the functions g_1, g_2, g_3 of two variables given by

$$g_1(n_1, n_2) = f(a, n_1, n_2)$$
$$g_2(n_1, n_2) = f(n_1, a, n_2)$$
$$g_3(n_1, n_2) = f(n_1, n_2, a)$$

is primitive recursive.

These are the functions defined from f by substituting the constant a for one of the variables of f.

The result of this problem can be generalized to show that, for any $k \geq 0$, if the function $f : \mathbb{N}^{k+1} \longrightarrow \mathbb{N}$ is primitive recursive and a is a fixed natural number, then for each $i = 1, 2, \ldots, k+1$ the function g_i of k variables defined by

$$g_i(n_1, n_2, \ldots, n_k) = f(n_1, \ldots, n_{i-1}, a, n_i, \ldots, n_k)$$

is primitive recursive. We shall often make repeated use of this result to substitute for several of the variables of a primitive recursive function f by constants, and we shall describe a function constructed in this way as defined from f *using constants*. Thus, for example, the function h in the solution to Problem 1.2 was defined from the primitive recursive function mult using constants, and hence we can deduce immediately that h is primitive recursive.

The constant a has been substituted for the ith variable of f.

$h(n) = \text{mult}(a, n)$.

Another result that will prove useful is exemplified by the following problem.

Problem 1.4

Show that if $f : \mathbb{N}^2 \longmapsto \mathbb{N}$ is primitive recursive then so is $g : \mathbb{N} \longmapsto \mathbb{N}$ given by $g(n) = f(n, n)$.

The result of this problem can be generalized to show that if the function $f : \mathbb{N}^k \longrightarrow \mathbb{N}$ is primitive recursive then so is the function $g : \mathbb{N}^j \longrightarrow \mathbb{N}$, where $j < k$, obtained by equating one or more of the variables of f.

In future we shall often not make so much fuss about the details when showing that particular functions are primitive recursive. In particular, we shall generally not bother to point out where projection functions and constant functions are used. Other details may fall by the wayside too. For example, to show that a function $h : \mathbb{N}^2 \longrightarrow \mathbb{N}$ defined by primitive recursion is a primitive recursive function, we shall usually be content to observe that the function $f(n) = h(n, 0)$ is a primitive recursive function and that $h(n, m+1)$ can be expressed in terms of n, m, $h(n, m)$ and constants using functions already known to be primitive recursive, confident that we could supply the full details if necessary. Using this less formal approach, we would be content to justify that mult is primitive recursive just by defining it by

We include the details in margin notes for the remainder of this subsection, to try to help you to see which details may safely be omitted when demonstrating that a function is primitive recursive.

$$\text{mult}(n, 0) = 0,$$
$$\text{mult}(n, m+1) = \text{add}(\text{mult}(n, m), n),$$

and then remarking that it is obtained by primitive recursion from the primitive recursive function add. Similarly for the function $n \longmapsto an + b$ we would be content to write

Notice that we do not bother to remark that we use the constant function C_0 to show that $f(n) = \text{mult}(n, 0)$ is primitive recursive or that we use the projection functions U_3^3 and U_1^3 to show that $\text{add}(\text{mult}(n, m), n)$ is primitive recursive.

$$an + b = \text{add}(\text{mult}(a, n), b)$$

and remark that $n \longmapsto an + b$ is obtained by substitution from the primitive recursive functions add and mult using constants. In fact, we may sometimes not even bother to write down the expression $\text{add}(\text{mult}(a, n), b)$ and may simply make the remark based on the expression $an + b$. We follow this less fussy approach in the following examples.

Example 1.4 Exponentiation

By *exponentiation* we mean the function $\exp : (n, m) \longmapsto n^m$. We shall show that this function is primitive recursive.

Note that $n^0 = 1$ and that $n^{m+1} = n^m \times n$. Thus we have the equations

$$\exp(n, 0) = 1,$$
$$\exp(n, m + 1) = \text{mult}(\exp(n, m), n),$$

which define exp by primitive recursion from the primitive recursive function mult. Thus exponentiation is a primitive recursive function. ♦

More formally, exp is defined by primitive recursion from the functions

$$f(n) = C_1(n),$$
$$g(n, m, k) = \text{mult}(U_3^3(n, m, k), U_1^3(n, m, k)),$$

where C_1 is primitive recursive and g is primitive recursive since it is obtained by substitution from the primitive recursive functions mult, U_3^3 and U_1^3.

Example 1.5 Cut-off subtraction

Another natural operation to consider is subtraction. Here we run into the technical problem that we are working with the set $\mathbb{N}$ of natural numbers, which is not closed under subtraction. For example $2 - 5 \notin \mathbb{N}$. We get over this difficulty by defining a new function called *cut-off subtraction* for which we use the symbol $\dot{-}$ and which is defined so that $n \dot{-} m = 0$ whenever $n - m < 0$: that is, it is defined by

$$n \dot{-} m = \begin{cases} n - m, & \text{if } m \leq n, \\ 0, & \text{if } n < m. \end{cases}$$

We shall show that the function $(n, m) \longmapsto n \dot{-} m$ is primitive recursive.

Anticipating the use of primitive recursion to define the function, we investigate whether $n \dot{-} (m + 1)$ can be defined in terms of $n \dot{-} m$. After a bit of thought, we see that

$$n \dot{-} (m + 1) = \begin{cases} (n \dot{-} m) - 1, & \text{if } n \dot{-} m > 0, \\ 0, & \text{otherwise.} \end{cases}$$

Hence we can define cut-off subtraction in terms of the simpler-looking function which maps k to $k - 1$ when $k > 0$ and maps 0 to 0. We call this the *predecessor* function, written pred. If pred is primitive recursive, then we can use it to show cut-off subtraction to be primitive recursive. The equations

$$\text{pred}(0) = 0,$$
$$\text{pred}(n + 1) = n,$$

define this function by primitive recursion from functions already known to be primitive recursive. Hence pred is a primitive recursive function.

More formally, pred is defined by primitive recursion from the constant 0 and the function

$$g(n, m) = U_1^2(n, m),$$

where U_1^2 is primitive recursive.

Now we can define cut-off subtraction by the equations

$$n \dot{-} 0 = n,$$
$$n \dot{-} (m + 1) = \text{pred}(n \dot{-} m).$$

This is a definition by primitive recursion from the primitive recursive function pred. Hence we can deduce that cut-off subtraction is a primitive recursive function. ♦

More formally, $\dot{-}$ is defined by primitive recursion from the functions

$$f(n) = U_1^1(n)$$
$$g(n, m, k) = \text{pred}(U_3^3(n, m, k)),$$

where U_1^1 is primitive recursive and g is primitive recursive since it is obtained by substitution from the primitive recursive functions pred and U_3^3.

Note that when trying to show that some interesting function is primitive recursive, we typically have to decompose it somehow into simpler-looking functions and try to show that these are primitive recursive.

The following problems establish that some further important functions are primitive recursive.

Problem 1.5

(a) Does $(n \dot{-} m) + 1$ always equal $n \dot{-} (m \dot{-} 1)$, where n and m are natural numbers?

(b) The *maximum* function, written as max, is defined by

$$\max(n, m) = \begin{cases} n, & \text{if } m \leq n, \\ m, & \text{if } n < m. \end{cases}$$

Show that $\max(n, m) = (n \dot{-} m) + m$ for all natural numbers n and m. Hence deduce that max is primitive recursive.

(c) Use cut-off subtraction to show that the *minimum* function min defined by

$$\min(n, m) = \begin{cases} m, & \text{if } m \leq n, \\ n, & \text{if } n < m, \end{cases}$$

is primitive recursive.

Problem 1.6

Show that the *absolute difference* function adf defined by

$$\text{adf}(n, m) = |n - m|$$

is primitive recursive.

It is worth noting that, already, our catalogue of primitive recursive functions includes the basic primitive recursive functions zero, succ and U_m^k, the constant functions C_a and C_a^k, and the functions add, mult, $n \longmapsto an + b$, exp, $\dot{-}$, pred, max, min and adf.

1.2 Characteristic functions

In this subsection we shall develop some of the machinery needed to show when a *function defined by cases* is primitive recursive, and hence URM-computable. Here is a typical example of such a function:

There are other examples in Subsection 1.1, e.g. the max and min functions in Problem 1.5.

$$f(n) = \begin{cases} \frac{1}{2}n, & \text{if } n \text{ is even}, \\ 3n + 1, & \text{if } n \text{ is odd}. \end{cases}$$

This function has two cases: the case when n is even and the case when n is odd. It is computed using a different formula in each case. There is an algorithm for computing the values of f since, given n, there is an algorithm which tells us whether n is even or odd; then, depending on what this algorithm tells us, we either apply the algorithm which computes $\frac{1}{2}n$ or the algorithm which computes $3n + 1$.

Informally, then, it is pretty convincing that this function f is computable. We shall establish this more formally within the contexts of URM-computability and primitive recursive functions. Our aim will be to combine the different cases of f into a single formula using primitive recursive functions.

In order to achieve this aim, we shall first need to clarify what we mean by an algorithm which tells us whether a particular case holds, for example whether a given number n is even, since up to now we have thought of algorithms as computing values of functions and not as answering questions such as whether a number is even. This is done by means of the notion of a *characteristic function* of a set. This is the function which takes the value 1 at a number n if n is in the given set and the value 0 at n if n is not in the given set.

Definition 1.3 Characteristic function of a set

If A is a subset of $\mathbb{N}$, then the *characteristic function* of A, which we denote by χ_A, is the function $\chi_A : \mathbb{N} \longrightarrow \mathbb{N}$ given by

$$\chi_A(n) = \begin{cases} 1, & \text{if } n \in A, \\ 0, & \text{if } n \notin A. \end{cases}$$

χ is the Greek letter 'chi' (pronounced *kye*, so as to rhyme with 'why').

Testing whether n is even is the same as testing whether n is in the set E of even numbers. So the question of whether there is an algorithm which tests whether n is even can now be phrased in terms of asking whether there is an algorithm for computing the value at n of characteristic function χ_E of the set E.

We shall use the idea of a characteristic function to extend the terminology of URM-computability and primitive recursion from functions to sets of natural numbers.

Definition 1.4

A subset A of $\mathbb{N}$ is said to be a *URM-computable set* if its characteristic function χ_A is a URM-computable function.

A subset A of $\mathbb{N}$ is said to be a *primitive recursive set* if χ_A is a primitive recursive function.

Note that, since every primitive recursive function is a URM-computable function, it follows immediately that every primitive recursive set is also a URM-computable set.

Example 1.6

The subsets $\{0\}$ and $\mathbb{Z}^+ = \{n : n \text{ is a positive integer}\}$ of $\mathbb{N}$ are primitive recursive sets.

We observe that $1 \dot{-} n = 1$ if $n = 0$ and $1 \dot{-} n = 0$ if $n > 0$. Thus

$$\chi_{\{0\}}(n) = 1 \dot{-} n,$$

and so $\chi_{\{0\}}$ is primitive recursive as it is obtained by substitution from the primitive recursive function $\dot{-}$ using constants. Thus $\{0\}$ is a primitive recursive set.

Next we observe that if $n = 0$ then $\chi_{\{0\}}(\chi_{\{0\}}(n)) = \chi_{\{0\}}(1) = 0$, while if $n > 0$ then $\chi_{\{0\}}(\chi_{\{0\}}(n)) = \chi_{\{0\}}(0) = 1$. Thus

$$\chi_{\mathbb{Z}^+}(n) = \chi_{\{0\}}(\chi_{\{0\}}(n))$$

and so $\chi_{\mathbb{Z}^+}$ is primitive recursive as it is obtained by substitution from the primitive recursive function $\chi_{\{0\}}$. Thus the set $\mathbb{Z}^+$ is a primitive recursive set. ♦

The characteristic function of the set $\mathbb{Z}^+$ of positive integers is traditionally known as the *signum* function and denoted by sg. We shall denote the characteristic function of $\{0\}$ by $\overline{\text{sg}}$, usually called *signum-bar*. These functions will be much used henceforth. Let us note that

$$\text{sg}(n) = \begin{cases} 1, & \text{if } n > 0, \\ 0, & \text{if } n = 0, \end{cases}$$

$$\overline{\text{sg}}(n) = \begin{cases} 1, & \text{if } n = 0, \\ 0, & \text{if } n > 0. \end{cases}$$

Problem 1.7

Show that the set E of even natural numbers is primitive recursive.

The set $\mathbb{Z}^+$ considered in Example 1.6 is the complement of the set $\{0\}$. In general the *complement* of a set A of natural numbers is the set $\{n \in \mathbb{N} : n \notin A\}$ and we write this set as $\mathbb{N} \setminus A$. The argument we used in Example 1.6 extends to this general case.

> ***Theorem 1.2***
>
> If the subset A of $\mathbb{N}$ is primitive recursive, then so also is its complement $\mathbb{N} \setminus A$.

Proof

As $\chi_{\mathbb{N}\setminus A}(n) = 1$ if and only if $\chi_A(n) = 0$, we see that $\chi_{\mathbb{N}\setminus A}$ is given by

$$\chi_{\mathbb{N}\setminus A}(n) = \overline{\text{sg}}(\chi_A(n)).$$

Thus $\chi_{\mathbb{N}\setminus A}$ is obtained by substitution from the primitive recursive functions $\overline{\text{sg}}$ and χ_A. Hence $\chi_{\mathbb{N}\setminus A}$ is a primitive recursive function and so $\mathbb{N} \setminus A$ is a primitive recursive set. ■

Problem 1.8

Prove that if the sets A and B of natural numbers are primitive recursive then so also are the sets $A \cap B$ and $A \cup B$.

From Problem 1.8 we can deduce that the intersection or union of any finite number of primitive recursive sets is a primitive recursive set.

Now we can show that the function f described at the beginning of this subsection is primitive recursive.

Example 1.7

Let E be the set of even natural numbers. The function f defined by

$$f(n) = \begin{cases} \frac{1}{2}n, & \text{if } n \in E, \\ 3n+1, & \text{if } n \notin E, \end{cases}$$

is primitive recursive.

Let $g : \mathbb{N} \longrightarrow \mathbb{N}$ be the function defined by

$$g(n) = \begin{cases} \frac{1}{2}n, & \text{if } n \in E, \\ \frac{1}{2}(n-1), & \text{if } n \notin E. \end{cases}$$

In choosing the function g, we need to ensure that $g(n) = \frac{1}{2}n$ if $n \in E$ and that $g(n) \in \mathbb{N}$ if $n \notin E$. We also desire that g can, fairly easily, be demonstrated to be primitive recursive. The function g we have chosen satisfies these conditions, though there are other functions we could have chosen.

We observe that $g(n+1) = g(n)$ if $n+1$ is odd and $g(n+1) = g(n)+1$ if $n+1$ is even.

Thus, since $n+1 = \text{succ}(n)$, g can be defined by the equations

$$g(0) = 0,$$
$$g(n+1) = g(n) + \chi_E(\text{succ}(n)).$$

It follows that g is primitive recursive, since it is defined by primitive recursion from the constant 0 and the primitive recursive functions add, χ_E and succ.

Alternatively, we can choose not to use succ and write

$$g(0) = 0,$$
$$g(n+1) = g(n) + \chi_E(n+1),$$

and declare that g is primitive recursive since it is defined by primitive recursion from the constant 0 and the primitive recursive functions add and χ_E using constants.

Remember that, to show that g is primitive recursive, we need to express $g(n+1)$ in terms of $g(n)$ and n, so we have to justify the presence of the expression $n+1$ on the right-hand side here. Since we have chosen not to use succ, the justification is that the function $n \longmapsto n+1$ is primitive recursive since it is obtained by substitution from the primitive recursive function add using constants. (Thus add is used twice here in showing that g is primitive recursive.)

Also $h : n \longmapsto 3n+1$ is primitive recursive because it is a linear function (Problem 1.2).

Now f may be defined by the single formula

$$f(n) = g(n)\chi_E(n) + h(n)\,\overline{\mathrm{sg}}(\chi_E(n)).$$

It is easily checked that this gives the correct values:

- if n is even, $\chi_E(n) = 1$ and $\overline{\mathrm{sg}}(\chi_E(n)) = 0$ so that $f(n) = g(n) = \frac{1}{2}n$;
- if n is odd, $\chi_E(n) = 0$ and $\overline{\mathrm{sg}}(\chi_E(n)) = 1$ so that $f(n) = h(n) = 3n+1$.

Since f is obtained by substitution from the primitive recursive functions add, mult, g, χ_E, h and $\overline{\mathrm{sg}}$, it is a primitive recursive function. ♦

The method we have used to show that the function f in Example 1.7 is primitive recursive can be generalized to cover other functions of one or more variables defined by cases. However, before doing this in the next subsection, we generalize the idea of a characteristic function.

You will have noticed that the cases in the original definition of the function f, given at the start of this subsection, weren't described in terms of whether n was in the set E of even numbers, but in terms of whether n was even. When defining a function of one variable by cases, it is fairly normal to express the cases in terms of a property of the input n, such as 'n is even' or 'n is prime', rather than in terms of the corresponding set (respectively 'n is in the set of even numbers' and 'n is in the set of primes'). In general, every property of natural numbers corresponds to a subset of $\mathbb{N}$. If P is a property of natural numbers, we write $P(n)$ to indicate that a number n has this property and $\{n : P(n)\}$ for the set of numbers with this property. An algorithm for testing whether n has the property P then corresponds to an algorithm for computing $\chi_A(n)$ where A is the set $\{n : P(n)\}$.

In the next subsection we shall look at the definition by cases of functions of more than one variable, where the cases are described by properties involving more than one variable. For example, we can consider the property of ordered pairs of numbers (n, m) that n is a divisor of m. A property involving ordered pairs of numbers is usually called a *relation*. Another example is the property that $n < m$, normally referred to as the $<$ relation (and pronounced 'the less than relation'). Given such a relation R, we write either nRm or $R(n, m)$ to signify that the ordered pair (n, m) has the relevant property. This is analogous with the situation for functions: for particular functions such as addition or multiplication, we usually write $n + m$ or $n \times m$, with the symbol for the function written between the variables, whereas we use notation such as $f(n, m)$ for a general function of two variables. Likewise we use notation such as $n \mid m$ or $n < m$ for particular relations between numbers, but notation such as $R(n, m)$ in the general case. As in the one-dimensional case, we can write $\{(n, m) : R(n, m)\}$ for the set of ordered pairs in $\mathbb{N}^2$ such that the relation R holds. We could then extend the definition of a characteristic function of a set to subsets A of $\mathbb{N}^2$, and an algorithm for testing whether the relation R holds for the ordered pair (n, m) would correspond to an algorithm for computing $\chi_A(n, m)$ where A is the set $\{(n, m) : R(n, m)\}$.

Recall, from Number Theory, *Unit 1*, that $n \mid m$ stands for 'n is a divisor of m'.

However, it simplifies matters if, instead of talking in terms of characteristic functions of sets, we define the characteristic function of a relation. This generalization of the idea of a characteristic function to relations enables us to focus directly on whether an ordered pair of natural numbers has the relevant property, rather than on whether that pair of numbers is in a particular set.

Definition 1.5 Characteristic function of a relation on $\mathbb{N}^2$

If R is a relation involving ordered pairs of natural numbers, its *characteristic function* is the function $\chi_R : \mathbb{N}^2 \longrightarrow \mathbb{N}$ given by

$$\chi_R(n,m) = \begin{cases} 1, & \text{if } R(n,m), \\ 0, & \text{otherwise.} \end{cases}$$

The relation R is said to be a *URM-computable relation* if its characteristic function χ_R is a URM-computable function.

The relation R is said to be a *primitive recursive relation* if its characteristic function χ_R is a primitive recursive function.

Example 1.8

The relation eq of equality between natural numbers is primitive recursive.

Of course we intend $eq(n,m)$ to hold if and only if $n = m$. We observe first that if $n = m$ then $|n - m|$ equals zero, but if $n \neq m$ then $|n - m|$ is positive. Therefore we can see that the characteristic function of eq is given by

$$\chi_{eq}(n,m) = \overline{\text{sg}}(\text{adf}(n,m)).$$

Since this function is defined by substitution from the primitive recursive functions $\overline{\text{sg}}$ and adf, it follows that χ_{eq} is also a primitive recursive function. Therefore eq is a primitive recursive relation. ♦

Recall from Problem 1.6 that the function adf, defined by $\text{adf}(n,m) = |n - m|$, is primitive recursive.

Problem 1.9

Prove that the relations $<$ and $\leq$ are primitive recursive.

Since $a > b$ if and only if $b < a$ and since $a \geq b$ if and only if $b \leq a$, we can deduce immediately from Problem 1.9 that the relations $>$ and $\geq$ are primitive recursive too.

The idea of a relation between two numbers can be extended to relations between three or more numbers. For example, the relation 'a is the arithmetic mean of b and c' relates three numbers. We use the notation $R(n_1, n_2, n_3)$ for a general relation involving an ordered triple of natural numbers and $R(n_1, n_2, \ldots, n_k)$ for a relation involving an ordered k-tuple of natural numbers. We refer to a relation involving an ordered k-tuple of natural numbers as a *k-place relation*. Our notions of characteristic function and primitive recursiveness in Definition 1.5 can be extended to such relations in the obvious way.

Definition 1.6 Characteristic function of a relation on $\mathbb{N}^k$

If R is a k-place relation, its *characteristic function* is the function $\chi_R : \mathbb{N}^k \longrightarrow \mathbb{N}$ given by

$$\chi_R(n_1, n_2, \ldots, n_k) = \begin{cases} 1, & \text{if } R(n_1, n_2, \ldots, n_k), \\ 0, & \text{otherwise.} \end{cases}$$

The relation R is said to be a *URM-computable relation* if its characteristic function χ_R is a URM-computable function.

The relation R is said to be a *primitive recursive relation* if its characteristic function χ_R is a primitive recursive function.

If $k = 2$ this corresponds to Definition 1.5. When $k = 1$ we have the characteristic function of the one-place relation, or property, $R(n_1)$. (For example, $R(n_1)$ could be the property 'n_1 is even'.) In this case $\chi_R(n_1) = \chi_A(n_1)$ where $A = \{n_1 : R(n_1)\}$.

Example 1.9

The relation R defined by

$$R(n_1, n_2, n_3) \iff n_1 \text{ is the arithmetic mean of } n_2 \text{ and } n_3$$

is primitive recursive.

We use the symbol $\iff$ as an abbreviation for 'if and only if'.

Since $R(n_1, n_2, n_3)$ if and only if $2n_1 = n_2 + n_3$, the characteristic function of R is given by

$$\chi_R(n_1, n_2, n_3) = \chi_{eq}(2n_1, n_2 + n_3).$$

Thus χ_R is obtained by substitution from the primitive recursive functions χ_{eq}, mult and add using constants. Therefore χ_R is a primitive recursive function. Hence R is a primitive recursive relation. ♦

Problem 1.10

Prove that if the k-place relations R and S are primitive recursive then so also are the k-place relations T, U and V defined, for all $(n_1, n_2, \ldots, n_k) \in \mathbb{N}^k$, by

$$T(n_1, n_2, \ldots, n_k) \iff \text{not } R(n_1, n_2, \ldots, n_k),$$
$$U(n_1, n_2, \ldots, n_k) \iff R(n_1, n_2, \ldots, n_k) \text{ and } S(n_1, n_2, \ldots, n_k),$$
$$V(n_1, n_2, \ldots, n_k) \iff R(n_1, n_2, \ldots, n_k) \text{ or } S(n_1, n_2, \ldots, n_k).$$

Problem 1.11

Show that the relation R defined by

$$R(n_1, n_2, n_3) \iff n_1 \leq n_2 \leq n_3$$

is primitive recursive. *Hint*: What is the characteristic function of the relation $n_1 \leq n_2$ regarded as a function of the three variables n_1, n_2, n_3?

The results in Problem 1.10 will prove very useful later.

Also useful later will be the analogue for relations of the result that a function obtained using constants from a primitive recursive function is primitive recursive. Thus if R is a primitive recursive $(k+1)$-place relation and a is a fixed natural number then, for each $i = 1, 2 \ldots, k+1$, the k-place relation S_i given by

$$S_i(n_1, n_2, \ldots, n_k) = R(n_1, \ldots, n_{i-1}, a, n_i, \ldots, n_k)$$

is primitive recursive. This result follows immediately from the corresponding result for functions using the characteristic functions χ_R and χ_{S_i}. We shall refer to a relation obtained in this way as being defined from R *using constants.*

1.3 Definition by cases

In this subsection we shall see how we can generalize the ideas in Example 1.7 so that we can show that a variety of functions defined by cases are primitive recursive. Before we can do this, however, we need to look again briefly at the definition of primitive recursion (*Unit 1*, Definition 3.3).

This definition prescribes that recursion is over the last variable: that is, to define a function $h : \mathbb{N}^{k+1} \longrightarrow \mathbb{N}$ by primitive recursion we need to specify $h(n_1, n_2, \ldots, n_k, 0)$ and to express $h(n_1, n_2, \ldots, n_k, n+1)$ in terms of $n_1, n_2, \ldots, n_k$, n and $h(n_1, n_2, \ldots, n_k, n)$. This is not always the most convenient way to proceed. Sometimes we might like to take a variable other than the last as the recursion variable. The next problem hints at a way round this difficulty.

For instance, if we wanted to show that $h(n, m) = m^n$ was primitive recursive by building it up from the function mult, we would much rather define $h(n+1, m)$ in terms of n, m and $h(n, m)$ (as $\text{mult}(m, h(n, m))$) than define $h(n, m+1)$ from these — you just try it!

Problem 1.12

Let $f : \mathbb{N}^3 \longrightarrow \mathbb{N}$ be a primitive recursive function and let $h : \mathbb{N}^3 \longrightarrow \mathbb{N}$ be defined by

$$h(n_1, n_2, n_3) = f(n_3, n_1, n_2).$$

Prove that h is a primitive recursive function.

Of course, the result of Problem 1.12 can be generalized. If $f : \mathbb{N}^k \longrightarrow \mathbb{N}$ is a primitive recursive function and σ is a permutation of $\{1, 2, \ldots, k\}$, then the function $h : \mathbb{N}^k \longrightarrow \mathbb{N}$ given by

$$h(n_1, n_2, \ldots, n_k) = f(n_{\sigma(1)}, n_{\sigma(2)}, \ldots, n_{\sigma(k)})$$

is primitive recursive, since

$$n_{\sigma(j)} = U^k_{\sigma(j)}(n_1, n_2, \ldots, n_k)$$

for $j = 1, 2, \ldots, k$ and we can thus see that h is obtained by substitution from f and projection functions. It follows that if a function h can be obtained from known primitive recursive functions by primitive recursion where a variable other than the last is taken as the recursion variable, the function h is primitive recursive. We shall exploit this greater freedom in the next example.

Formally the *permutation* σ is a one–one and onto function, i.e. a one–one correspondence or bijection, from the set $\{1, 2, \ldots, k\}$ to itself. The effect of σ is to rearrange the order of the variables to which f applies. (We define the terms 'one–one', 'onto', etc. at the start of Subsection 2.1.)

σ is the Greek letter 'sigma'.

Example 1.10

From the Division Algorithm (Number Theory, *Unit 1*) we know that for any two natural numbers n and m, where $m > 0$, there exist unique natural numbers q and r such that

$$n = mq + r, \quad \text{where } 0 \leq r < m.$$

The number q is called the *quotient* when n is divided by m and the number r is called the *remainder* when n is divided by m. We are going to define a *quotient function* $\text{quot} : \mathbb{N}^2 \longrightarrow \mathbb{N}$ such that $\text{quot}(n, m)$ is the quotient when n is divided by m if $m \neq 0$ and a *remainder function* $\text{rem} : \mathbb{N}^2 \longrightarrow \mathbb{N}$ such that $\text{rem}(n, m)$ is the remainder when n is divided by m if $m \neq 0$. To complete the definitions, we need to specify the values of $\text{quot}(n, m)$ and $\text{rem}(n, m)$ when $m = 0$ and we can do this arbitrarily. Thus we make the definitions

$$\text{quot}(n, m) = \begin{cases} \text{the quotient when } n \text{ is divided by } m, & \text{if } m \neq 0, \\ 0, & \text{if } m = 0, \end{cases}$$

$$\text{rem}(n, m) = \begin{cases} \text{the remainder when } n \text{ is divided by } m, & \text{if } m \neq 0, \\ 0, & \text{if } m = 0. \end{cases}$$

We shall show that the functions $\text{quot} : \mathbb{N}^2 \longrightarrow \mathbb{N}$ and $\text{rem} : \mathbb{N}^2 \longrightarrow \mathbb{N}$ are primitive recursive.

We shall consider the function rem first because we shall use the fact that rem is primitive recursive in the proof that quot is primitive recursive. To see how recursion works for rem when $m \neq 0$, it is helpful to look at a particular numerical example. The following table gives the values of $\mathrm{rem}(n, 3)$ for $0 \leq n \leq 6$.

n	$\mathrm{rem}(n,3)$
0	0
1	1
2	2
3	0
4	1
5	2
6	0

We see from this that, as the value of n increases by 1, the value of the remainder increases by 1 except when the remainder is 2, that is, one less than the divisor, when the value of the remainder drops back to zero. We can sum this up by saying that when $m \neq 0$ we have

$$\mathrm{rem}(n+1, m) = \begin{cases} 0, & \text{if } \mathrm{rem}(n,m) = m-1, \\ \mathrm{rem}(n,m)+1, & \text{if } \mathrm{rem}(n,m) \neq m-1. \end{cases}$$

Thus taking into account our decision that $\mathrm{rem}(n, m) = 0$ when $m = 0$, we see that we can define the function rem by the equations

$$\mathrm{rem}(0, m) = 0,$$

$$\mathrm{rem}(n+1, m) = \begin{cases} 0, & \text{if } m = 0 \text{ or } \mathrm{rem}(n,m) = m-1, \\ \mathrm{rem}(n,m)+1, & \text{otherwise.} \end{cases}$$

Since $\mathrm{sg}(m) = 1$ if and only if $m \neq 0$ and $\overline{\mathrm{sg}}(\chi_{eq}(\mathrm{rem}(n, m), m \dot{-} 1)) = 1$ if and only if $\mathrm{rem}(n, m) \neq m \dot{-} 1$, we see that

$$\mathrm{sg}(m)\,\overline{\mathrm{sg}}(\chi_{eq}(\mathrm{rem}(n, m), m \dot{-} 1))$$

is equal to 1 if and only if $m > 0$ and $\mathrm{rem}(n, m) \neq m \dot{-} 1$ and it is equal to 0 otherwise (i.e. if $m = 0$ or $\mathrm{rem}(n, m) = m \dot{-} 1$). Hence we have

$$\mathrm{rem}(n+1, m) = (\mathrm{rem}(n, m) + 1)\,\mathrm{sg}(m)\,\overline{\mathrm{sg}}(\chi_{eq}(\mathrm{rem}(n, m), m \dot{-} 1)).$$

Thus we have

$$\mathrm{rem}(0, m) = 0,$$
$$\mathrm{rem}(n+1, m) = (\mathrm{rem}(n, m) + 1)\,\mathrm{sg}(m)\,\overline{\mathrm{sg}}(\chi_{eq}(\mathrm{rem}(n, m), m \dot{-} 1)),$$

so that rem is obtained by primitive recursion (over the first variable) from the primitive recursive functions mult, add, sg, $\overline{\mathrm{sg}}$, χ_{eq} and $\dot{-}$ using constants. It follows that the remainder function rem is primitive recursive.

Now we can show that the quotient function quot is primitive recursive. Notice that if $m \neq 0$ and $n = mq + r$ where $0 \leq r < m$ then $\dfrac{n}{m} = q + \dfrac{r}{m}$ where $0 \leq \dfrac{r}{m} < 1$. Thus if $m > 0$ then $\mathrm{quot}(n, m)$ is the integer part $\left[\dfrac{n}{m}\right]$ of $\dfrac{n}{m}$. So we have

The *integer part* of a real number r, often referred to as the *floor* of r, is the largest integer not greater than r. It is denoted by $[r]$.

$$\mathrm{quot}(n, m) = \begin{cases} 0, & \text{if } m = 0, \\ \left[\dfrac{n}{m}\right], & \text{if } m \neq 0. \end{cases}$$

It can be seen that, for $m \neq 0$,

$$\left[\frac{n+1}{m}\right] = \left[\frac{n}{m}\right]$$

unless m is a divisor of $n+1$, in which case

$$\left[\frac{n+1}{m}\right] = \left[\frac{n}{m}\right] + 1.$$

Thus we see that, for $m \neq 0$,

$$\text{quot}(n+1, m) = \begin{cases} \text{quot}(n, m) + 1, & \text{if } \text{rem}(n+1, m) = 0, \\ \text{quot}(n, m), & \text{if } \text{rem}(n+1, m) \neq 0. \end{cases}$$

Note that

$$\overline{\text{sg}}(\text{rem}(n+1, m)) = \begin{cases} 1, & \text{if } \text{rem}(n+1, m) = 0, \\ 0, & \text{if } \text{rem}(n+1, m) \neq 0. \end{cases}$$

Hence the quotient function is defined by the equations

$$\text{quot}(0, m) = 0,$$

$$\text{quot}(n+1, m) = \text{sg}(m)(\text{quot}(n, m) + \overline{\text{sg}}(\text{rem}(n+1, m))).$$

The factor $\text{sg}(m)$ is needed to cover the case when $m = 0$.

Thus quot is obtained by primitive recursion (over the first variable) from the primitive recursive functions mult, sg, add, $\overline{\text{sg}}$ and rem using constants. Hence the quotient function quot is primitive recursive. ♦

Problem 1.13

The *binomial coefficient* $\binom{n}{r}$ is defined for all $n, r \in \mathbb{N}$ with $r \leq n$ by the formula

$$\binom{n}{r} = \frac{n!}{r!(n-r)!},$$

where $\binom{n}{r}$ is always a natural number.

To obtain a function which is defined for all $n, r \in \mathbb{N}$ and which equals the binomial coefficient when $r \leq n$, we define the function $C : \mathbb{N}^2 \longrightarrow \mathbb{N}$ by the formula

$$C(n, r) = \begin{cases} \binom{n}{r}, & \text{if } r \leq n, \\ 0, & \text{if } r > n. \end{cases}$$

Show that the function C is primitive recursive. *Hint:* First show that the factorial function $\text{fac} : n \longmapsto n!$ is primitive recursive.

Our argument in Example 1.10 to show that the function rem is primitive recursive involved converting the formula

$$\text{rem}(n+1, m) = \begin{cases} 0, & \text{if } m = 0 \text{ or } \text{rem}(n, m) = m \dot{-} 1, \\ \text{rem}(n, m) + 1, & \text{otherwise,} \end{cases}$$

which defines the value of $\text{rem}(n+1, m)$ according to cases, into a single equation

$$\text{rem}(n+1, m) = (\text{rem}(n, m) + 1)\,\text{sg}(m)\,\overline{\text{sg}}(\chi_{eq}(\text{rem}(n, m), m \dot{-} 1)).$$

We used a similar technique in showing that the function quot is primitive recursive in Example 1.10 and in showing that the function f is primitive recursive in Example 1.7. Now we generalize this idea.

The simplest case is where the function is from $\mathbb{N}$ to $\mathbb{N}$ and is defined in one of two ways according to whether a number is in a given set or not.

Theorem 1.3

If A is a primitive recursive set and $g : \mathbb{N} \longrightarrow \mathbb{N}$, $h : \mathbb{N} \longrightarrow \mathbb{N}$ are primitive recursive functions, then the function $f : \mathbb{N} \longrightarrow \mathbb{N}$ defined by

$$f(n) = \begin{cases} g(n), & \text{if } n \in A, \\ h(n), & \text{if } n \notin A, \end{cases}$$

is also primitive recursive.

Proof

The function f is given by the formula

$$f(n) = g(n)\chi_A(n) + h(n)\,\overline{\text{sg}}(\chi_A(n)),$$

for if $n \in A$, $\chi_A(n) = 1$ and $\overline{\text{sg}}(\chi_A(n)) = 0$, so the right-hand side is equal to $g(n)$, and if $n \notin A$, $\chi_A(n) = 0$ and $\overline{\text{sg}}(\chi_A(n)) = 1$, so the right-hand side is equal to $h(n)$. Hence, as add, mult, g, χ_A, h and $\overline{\text{sg}}$ are primitive recursive functions, and as f is obtained from these functions by substitution, f is also primitive recursive. ■

Recall, from the proof of Theorem 1.2, that $\chi_{\mathbb{N}\setminus A}(n) = \overline{\text{sg}}(\chi_A(n))$.

As we mentioned at the start of Subsection 1.2, we call the formula which defines f in Theorem 1.3 a *definition by cases* because $f(n)$ is equal to either $g(n)$ or $h(n)$ depending on which of the cases $n \in A$ and $n \notin A$ holds. Clearly one or other, but not both, of these cases must hold so this gives us an unambiguous value for $f(n)$ for each natural number n. This is a very important technique and we wish to extend it to more than two cases.

Suppose we have k sets of natural numbers $A_1, A_2, \ldots, A_k$ and k functions $g_1, g_2, \ldots, g_k$ from $\mathbb{N}$ to $\mathbb{N}$, and that we wish to define a function f from $\mathbb{N}$ to $\mathbb{N}$ such that $f(n) = g_1(n)$ if $n \in A_1$, $f(n) = g_2(n)$ if $n \in A_2, \ldots, f(n) = g_k(n)$ if $n \in A_k$. Then to ensure that the domain of f is $\mathbb{N}$ we must require that every natural number n is in one of the sets $A_1, A_2, \ldots, A_k$, while to ensure that the values of f are unambiguously defined we require that no natural number is in more than one of the sets $A_1, A_2, \ldots, A_k$. In other words every natural number must be in *exactly one* of the sets $A_1, A_2, \ldots, A_k$. This is equivalent to saying that the sets $A_1, A_2, \ldots, A_k$ form a *partition* of $\mathbb{N}$, meaning that $A_1 \cup A_2 \cup \cdots \cup A_k = \mathbb{N}$ and $A_r \cap A_s = \emptyset$ whenever $r \neq s$. Note that if A is a subset of $\mathbb{N}$ then A and $\mathbb{N} \setminus A$ form a partition of $\mathbb{N}$, so our next result generalizes Theorem 1.3.

For f to be a function, we need just the one value for $f(n)$. If n was in more than one of the A_is, we could end up with conflicting values for $f(n)$, and hence an ambiguity.

Theorem 1.4

Suppose that $A_1, A_2, \ldots, A_k$ are primitive recursive sets which form a partition of $\mathbb{N}$ and that $g_1, g_2, \ldots, g_k$ are primitive recursive functions from $\mathbb{N}$ to $\mathbb{N}$. Then the function $f : \mathbb{N} \longrightarrow \mathbb{N}$ defined by

$$f(n) = \begin{cases} g_1(n), & \text{if } n \in A_1, \\ g_2(n), & \text{if } n \in A_2, \\ \quad\vdots & \quad\vdots \\ g_k(n), & \text{if } n \in A_k, \end{cases}$$

is also primitive recursive.

Proof

We have

$$f(n) = g_1(n)\chi_{A_1}(n) + g_2(n)\chi_{A_2}(n) + \cdots + g_k(n)\chi_{A_k}(n);$$

for if $n \in \mathbb{N}$ there is a unique r such that $n \in A_r$; then $\chi_{A_r}(n) = 1$ and $\chi_{A_s}(n) = 0$ for $s \neq r$, and the value of the right-hand side is $g_r(n)$ as required. Since the sets $A_1, A_2, \ldots, A_k$ are primitive recursive, the functions $\chi_{A_1}, \chi_{A_2}, \cdots, \chi_{A_k}$ are primitive recursive. Hence f is obtained by substitution from the primitive recursive functions add, g_1, $g_2, \ldots$, g_k and χ_{A_1}, $\chi_{A_2}, \cdots$, χ_{A_k}, and so f is primitive recursive. ■

We can extend everything we have said about definition by cases to functions of more than one variable. All we need to do is to replace the sets $A_1, A_2 \ldots, A_k$ by appropriate relations. In this case instead of talking about partitions we say that the relations involved are mutually exclusive and exhaustive. To be precise, if $R_1, R_2, \ldots, R_k$ are k l-place relations then we say that these relations are *mutually exclusive and exhaustive* if, for all $(n_1, n_2, \ldots, n_l) \in \mathbb{N}^l$, precisely one of $R_1(n_1, n_2, \ldots, n_l), R_2(n_1, n_2, \ldots, n_l)$, $\ldots, R_k(n_1, n_2, \ldots, n_l)$ holds.

'Mutually exclusive' means that if $R_i(n_1, n_2, \ldots, n_l)$ holds then $R_j(n_1, n_2, \ldots, n_l)$ does not hold for $j \neq i$. 'Exhaustive' means that, for each $(n_1, n_2, \ldots, n_l) \in \mathbb{N}^l$, $R_i(n_1, n_2, \ldots, n_l)$ holds for at least one $i \in \{1, 2, \ldots, k\}$.

Theorem 1.5 Definition by cases

Suppose that $R_1, R_2, \ldots, R_k$ are primitive recursive l-place relations which are mutually exclusive and exhaustive and that $g_1, g_2, \ldots, g_k$ are primitive recursive functions from $\mathbb{N}^l$ to $\mathbb{N}$. Then the function $f : \mathbb{N}^l \longrightarrow \mathbb{N}$ defined by

$$f(n_1, n_2, \ldots, n_l) = \begin{cases} g_1(n_1, n_2, \ldots, n_l), & \text{if } R_1(n_1, n_2, \ldots, n_l), \\ g_2(n_1, n_2, \ldots, n_l), & \text{if } R_2(n_1, n_2, \ldots, n_l), \\ \quad \vdots & \quad \vdots \\ g_k(n_1, n_2, \ldots, n_l), & \text{if } R_k(n_1, n_2, \ldots, n_l), \end{cases}$$

is also primitive recursive.

Problem 1.14

Prove Theorem 1.5.

Problem 1.15

Let R_1 and R_2 be the two-place relations defined by

$$R_1(n, m) \iff \max(n, 2m) \geq 21,$$
$$R_2(n, m) \iff 3n + 4m = 43$$

and let $f : \mathbb{N}^2 \longrightarrow \mathbb{N}$ be the function defined by

$$f(n, m) = \begin{cases} g_1(n, m), & \text{if } R_1(n, m), \\ g_2(n, m), & \text{if } R_2(n, m), \\ g_3(n, m), & \text{otherwise}, \end{cases}$$

where g_1, g_2, g_3 are primitive recursive functions from $\mathbb{N}^2$ to $\mathbb{N}$. Show that f is a primitive recursive function.

The technique for dealing with the 'otherwise' case in Problem 1.15 can be generalized so that if $f : \mathbb{N}^l \longrightarrow \mathbb{N}$ is defined by

$$f(n_1, n_2, \ldots, n_l) = \begin{cases} g_1(n_1, n_2, \ldots, n_l), & \text{if } R_1(n_1, n_2, \ldots, n_l), \\ g_2(n_1, n_2, \ldots, n_l), & \text{if } R_2(n_1, n_2, \ldots, n_l), \\ \quad \vdots & \quad \vdots \\ g_{k-1}(n_1, n_2, \ldots, n_l), & \text{if } R_{k-1}(n_1, n_2, \ldots, n_l), \\ g_k(n_1, n_2, \ldots, n_l), & \text{otherwise,} \end{cases}$$

then the 'otherwise' case can be replaced by 'if $R_k(n_1, n_2, \ldots, n_l)$' where

$$R_k \iff \text{not } R_1 \text{ and not } R_2 \text{ and } \ldots \text{ and not } R_{k-1},$$

and R_k is primitive recursive if R_1, $R_2, \ldots$, R_{k-1} are (by Problem 1.10).

2 INFINITE SETS

We have seen that primitive recursion is a powerful way to generate functions and thus we have found many examples of primitive recursive functions. Since all primitive recursive functions are URM-computable, we have also shown how to generate a large stock of URM-computable functions. This supports our claim that the URM programming language is much more powerful than it seems at first sight.

It is now natural to ask whether we need to go beyond primitive recursion. There are two questions that we might ask.

- Are there any functions of any sort that are not primitive recursive?
- Are there any URM-computable functions that are not primitive recursive?

In this section we are going to describe Cantor's theory of countably infinite sets. This will readily show us that there are functions which are not primitive recursive. It will also lead us to conclude, in *Unit 3*, that there are URM-computable functions which are not primitive recursive.

Georg Cantor (1843–1918) founded the study of infinite numbers and sets. His work provoked great interest and controversy in the foundations of mathematics.
(Photo © Science Photo Library)

2.1 Cantor's theory of countably infinite sets

A central idea in Cantor's theory is that of a *one–one correspondence*, which is a concept with which you may be familiar. Let $f : A \longrightarrow B$ be a function. The function f is said to be *one–one* if, for all x_1, x_2 in A, whenever $f(x_1) = f(x_2)$ then $x_1 = x_2$. Thus if f is a one–one function then, for each y in B, there is at most one x in A such that $f(x) = y$. The function f is said to be *onto* if $f(A) = B$: that is, for each y in B there is at least one x in A such that $f(x) = y$. The function f is said to be a *one–one correspondence* if it is both one–one and onto. Thus if $f : A \longrightarrow B$ is a one–one correspondence then, for each y in B, there is exactly one x in A such that $f(x) = y$, and f does indeed set up a correspondence between the elements of A and B. Moreover, if $f : A \longrightarrow B$ is a one–one correspondence then the function $f^{-1} : B \longrightarrow A$ given by

$$f^{-1}(y) = x \quad \text{whenever} \quad f(x) = y$$

is well-defined and is a one–one correspondence which we call the *inverse* of f. We shall sometimes use the notation $x \longleftrightarrow y$ to indicate that x is mapped to y by a one–one correspondence and so y is mapped to x by its inverse.

A one–one correspondence is also often called a *bijection*.

In mathematics we meet many examples of infinite sets. In the Mathematical Logic part of M381, the set $\mathbb{N}$ of natural numbers plays an important role, and in Number Theory you have met the set $\mathbb{Z}$ of all integers and the set $\mathbb{Z}^+$ of positive integers. Elsewhere you may have met the set $\mathbb{Q}$ of rational numbers, the set $\mathbb{R}$ of real numbers, the set $\mathbb{R}^2$ of points in the plane and the set $\mathbb{C}$ of complex numbers. At first it might seem that all we can say about the sizes of these sets is that they are infinite. However, by generalizing from the way in which we count finite sets, Cantor was able to distinguish different infinities.

If we want to know whether there are as many dots as crosses here

$$\bullet\bullet\bullet\bullet\bullet\bullet\bullet\bullet\bullet\bullet\bullet\bullet \qquad \times\times\times\times\times\times\times\times\times\times\times\times$$

we can do this by lining the dots up with the crosses and checking that there is a one–one correspondence between the elements of the two sets:

$$\begin{array}{cccccccccccc} \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow \\ \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \end{array}$$

Another way to check that we have equal numbers of dots and crosses would be to count them, but this amounts to pretty much the same thing, as it involves putting each set into a one–one correspondence with the integers from 1 to 12, and hence in one–one correspondence with each other:

$$\begin{array}{cccccccccccc} \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\ \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow \\ \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \end{array}$$

Cantor's idea was to extend this to infinite sets. He said that, in general, two sets contain the same number of elements if there is a one–one correspondence between them. We shall look in detail only at sets which can be put in one–one correspondence with the set $\mathbb{N}$ of natural numbers.

The elements of $\mathbb{N}$ can be enumerated in a single list:

$$0,\ 1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7,\ \ldots$$

There is a crucial difference between this and the list of natural numbers we used above to count the number of dots and crosses. That list was *finite*. It came to an end after we reached the number 12. On the other hand the list of elements of $\mathbb{N}$ never ends; $\mathbb{N}$ is an *infinite* set. However, Cantor had the confidence to extend to $\mathbb{N}$ ideas of counting derived from our experience with finite sets. Here is Cantor's key definition.

Definition 2.1 Countably infinite set

A set A is said to be *countably infinite* if there is a one–one correspondence between $\mathbb{N}$ and A.

Note that, in some texts, the terms *denumerable* or *countable* are used instead of *countably infinite*.

Example 2.1

The set E of even natural numbers and the set O of odd natural numbers are both countably infinite since we have the following one–one correspondences between $\mathbb{N}$ and E and between $\mathbb{N}$ and O.

$$\begin{array}{cccccccccc} E & 0 & 2 & 4 & 6 & 8 & \ldots & 2n & \ldots \\ & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \ldots & \updownarrow & \ldots \\ \mathbb{N} & 0 & 1 & 2 & 3 & 4 & \ldots & n & \ldots \\ & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \ldots & \updownarrow & \ldots \\ O & 1 & 3 & 5 & 7 & 9 & \ldots & 2n+1 & \ldots \end{array}$$

♦

In fact, every infinite subset of $\mathbb{N}$ is countably infinite. For suppose X is an infinite subset of $\mathbb{N}$. We can list the elements of X in ascending order and this gives the required one–one correspondence with $\mathbb{N}$ as follows.

$$\begin{array}{ccccccccccc} X & x_0 & < & x_1 & < & x_2 & < & \ldots & < & x_n & < & \ldots \\ & \updownarrow & & \updownarrow & & \updownarrow & & \ldots & & \updownarrow & & \ldots \\ \mathbb{N} & 0 & & 1 & & 2 & & \ldots & & n & & \ldots \end{array}$$

In the case of the set E we have a very simple formula, namely $n \longleftrightarrow 2n$, which defines a one–one correspondence between $\mathbb{N}$ and E. In general, we cannot expect there to be a straightforward formula. For example, while the set of prime numbers is countably infinite, there is no simple formula which defines a one–one correspondence between $\mathbb{N}$ and the set of prime numbers.

If a set X is countably infinite then there is a one–one correspondence between $\mathbb{N}$ and X, and its inverse is a one–one correspondence between X and $\mathbb{N}$. Conversely, if there is a one–one correspondence between X and $\mathbb{N}$ then its inverse is a one–one correspondence between $\mathbb{N}$ and X, and so X is countably infinite. In fact it follows from our remark about infinite subsets of $\mathbb{N}$ that when X is infinite we need only find a one–one function mapping X *into* $\mathbb{N}$ in order to be able to deduce that X is countably infinite.

Theorem 2.1

Let X be an infinite set and suppose that there is a one–one function $f : X \longrightarrow \mathbb{N}$. Then X is countably infinite.

Proof

Put $Y = \{f(x) : x \in X\}$. Then, since f is one–one, Y is an infinite subset of $\mathbb{N}$. Thus Y is countably infinite and so there is a one–one correspondence $g : Y \longrightarrow \mathbb{N}$. The function $x \longmapsto g(f(x))$ from X to $\mathbb{N}$ is both one–one and onto: that is, it is a one–one correspondence. Hence X is countably infinite. ■

We can also generalize our observation that every infinite subset of $\mathbb{N}$ is countably infinite as follows.

Theorem 2.2

Let X be a countably infinite set and let Y be an infinite subset of X. Then Y is countably infinite.

Proof

Since X is countably infinite, there is a one–one correspondence $f : X \longrightarrow \mathbb{N}$. The *restriction* of f to Y (that is, the function normally written as $f|_Y : Y \longrightarrow \mathbb{N}$ and defined by $f|_Y(y) = f(y)$ for $y \in Y$) is a one–one function. Hence, as Y is infinite, Y is countably infinite by Theorem 2.1 ■

The usefulness of Theorem 2.1 will be illustrated by examples to follow.

Example 2.2

The set $\mathbb{N}^2$ of all pairs of natural numbers is countably infinite.

Every natural number greater than 1 has a unique prime decomposition as a product of positive integer powers of its distinct prime divisors (Fundamental Theorem of Arithmetic, Number Theory, *Unit 2*). We can deduce that, if a number can be written in the form 2^n3^m for $n, m \in \mathbb{N}$, it can be done so in just one way.

The numbers $1 = 2^0 3^0$, $2^n = 2^n 3^0$ for $n > 0$, $3^m = 2^0 3^m$ for $m > 0$ and $2^n 3^m$ for $n, m > 0$ are all distinct.

Now consider the function $f : \mathbb{N}^2 \longrightarrow \mathbb{N}$ defined by $f(n, m) = 2^n 3^m$. This is a one–one function. For suppose that $m, n, r, s \in \mathbb{N}$ and $f(n, m) = f(r, s)$. Then $2^n 3^m = 2^r 3^s$ so that $n = r$ and $m = s$ by the remark above: that is, $(n, m) = (r, s)$.

Since $f : \mathbb{N}^2 \longrightarrow \mathbb{N}$ is a one–one function and $\mathbb{N}^2$ is infinite, it follows from Theorem 2.1 that $\mathbb{N}^2$ is countably infinite. ♦

The function f in Example 2.2 and similar functions f_k which you will meet in the next example will play an important role in representing a URM program by a code number, so in the next two problems we ask you to do some computations involving these functions.

Problem 2.1

Let f be the function defined in Example 2.2.

(a) Evaluate $f(3, 2)$ and $f(2, 3)$.

(b) In each of the following cases, find the values of n and m:

(i) $f(n, m) = 12$; (ii) $f(n, m) = 16$; (iii) $f(n, m) = 216$.

We shall make frequent use of the prime decomposition of integers and the Fundamental Theorem of Arithmetic (as in Example 2.2). Also we shall use Euclid's Theorem (Number Theory, *Unit 2*) that there are infinitely many primes. This justifies the use of the first k primes, for k an arbitrary positive integer, in the next example.

Example 2.3

For $k > 1$, the set $\mathbb{N}^k$ of all ordered k-tuples of natural numbers is countably infinite.

We can easily generalize the argument in Example 2.2. All we need to do is to replace the first two prime numbers 2 and 3 by the first k prime numbers. We let these be $p_1, p_2, \ldots, p_k$. Thus $p_1 = 2$, $p_2 = 3$, $p_3 = 5$ and so on.

Now consider the function $f_k : \mathbb{N}^k \longrightarrow \mathbb{N}$ defined by

$$f_k(n_1, n_2, \ldots, n_k) = p_1^{n_1} p_2^{n_2} \ldots p_k^{n_k}.$$

It follows from the Fundamental Theorem of Arithmetic that f_k is a one–one function. Since $\mathbb{N}^k$ is infinite, it follows from Theorem 2.1 that $\mathbb{N}^k$ is countably infinite. ♦

For each $(n_1, n_2, \ldots, n_k)$, including the cases where some of the n_i are 0, a distinct product $p_1^{n_1} p_2^{n_2} \ldots p_k^{n_k}$ is obtained.

Problem 2.2

For $k > 1$, let f_k be the function defined in Example 2.3.

(a) Evaluate $f_3(3, 2, 1)$ and $f_4(2, 1, 2, 1)$.

(b) (i) Find the values of n_1, n_2 and n_3 if $f_3(n_1, n_2, n_3) = 240$.

(ii) Find the values of n_1, n_2, n_3 and n_4 if $f_4(n_1, n_2, n_3, n_4) = 100$.

The next theorem uses a simple but useful idea.

> ***Theorem 2.3***
>
> If the sets A and B are both countably infinite then so also is the set $A \cup B$.

Proof

Suppose that A and B are both countably infinite. Then there are one–one functions $f : A \longrightarrow \mathbb{N}$ and $g : B \longrightarrow \mathbb{N}$. Now we define $h : A \cup B \longrightarrow \mathbb{N}$ by

$$h(x) = \begin{cases} 2f(x), & \text{if } x \in A, \\ 2g(x) + 1, & \text{if } x \in B \text{ and } x \notin A. \end{cases}$$

Since, for each $x \in A$, $h(x)$ is a multiple of 2 whereas, for $x \in B$ and $x \notin A$, $h(x)$ is not a multiple of 2, it is easily seen that h is a one–one function. Since $A \cup B$ is infinite, it follows from Theorem 2.1 that $A \cup B$ is countably infinite. ■

The technique of arranging the values of $h(x)$ to be even for $x \in A$ and odd otherwise, to ensure that h is a one–one function, will be useful later on.

Problem 2.3

Show that the set $\mathbb{Z}$ of integers is countably infinite.

The alternative method given in Solution 2.3 of showing that a set is countably infinite by listing its elements in a single sequence will be used in the next example.

In the Introduction to *Unit 1*, we mentioned that in the Mathematical Logic part of the course we shall be treating the formulas of mathematical logic as finite strings of symbols drawn from a finite set of symbols. Later in this part of the course it is going to prove useful to know that the set of all such strings is countably infinite. In the following example we shall see that this is so when the set of symbols is the English alphabet, so that the strings are the finite strings of letters from this alphabet: for example, *ab* is a two-letter string, *xaa* is a three-letter string, *uvuw* is a four-letter string, and so on.

Example 2.4

The set of finite strings of letters from the English alphabet is countably infinite.

We shall indicate, without giving precise details, how we can list the strings in a single sequence. The first 26 entries in the list are the 26 letters of the alphabet in their usual order. The next 676 (= 26 × 26) entries are the two-letter strings in dictionary order: that is,

$$aa, ab, \ldots, az, ba, \ldots bz, \ldots, za, \ldots, zz.$$

Next we list the three-letter strings in dictionary order: that is, $aaa, aab, \ldots, zzz$; and so on. This does list the strings in a single sequence. With a little work we can determine the kth entry in this list for each positive integer k (for example, the 1000th entry is the three-letter string *all*) and the position of any given string in the list can be determined. This list can be used to establish a one–one correspondence between the set of all finite strings and $\mathbb{N}$. Thus the set of all finite strings is countably infinite. ♦

We are now ready to prove that the set of URM programs is countably infinite. We do this in the next subsection. It will be seen that the method we have used in Example 2.3 to show that $\mathbb{N}^k$ is countably infinite, for each $k > 1$, plays a key role in the argument.

2.2 Coding URM programs

In this subsection we prove that the set of URM programs is countably infinite. In the process we shall represent every URM program P by a number from which we can recover the original program — this will be of immense importance later.

We begin by considering the instructions which make up a URM program. Recall that the URM instructions take the following forms:

$$Z(n), \quad S(n), \quad C(m,n), \quad J(m,n,q),$$

where m, n and q can be any positive integers. We let $\mathbf{I}$ be the set of all these URM instructions and we define a mapping $\beta : \mathbf{I} \longrightarrow \mathbb{N}$ as follows:

β is the Greek letter 'beta'.

$$\begin{aligned}
&\beta(Z(n)) = 6n - 3,\\
&\beta(S(n)) = 6n,\\
&\beta(C(m,n)) = 2^m 3^n + 1,\\
&\beta(J(m,n,q)) = 2^m 3^n 5^q + 2.
\end{aligned}$$

Note that $\beta(Z(n))$ is always a multiple of 3 but not a multiple of 6, $\beta(S(n))$ is always a multiple of 6, $\beta(C(m,n))$ always has remainder 1 when divided by 3 and $\beta(J(m,n,q))$ has remainder 2 when divided by 3.

Definition 2.2

For each URM instruction I, $\beta(I)$ is called the *code number* of I, or *code* of I for short. We say that $\beta(I)$ *codes* the instruction I.

We can use β to prove the following theorem.

Theorem 2.4

The set $\mathbf{I}$ of URM instructions is countably infinite.

Proof

We shall first show that $\beta : \mathbf{I} \longrightarrow \mathbb{N}$ is a one–one function.

From the remark following the definition of β, we can see that no two instructions of different types can have the same code number. Also it is clear that different Zero instructions and different Successor instructions have different code numbers. Suppose that $\beta(C(m,n)) = \beta(C(r,s))$. Then $2^m 3^n = 2^r 3^s$ so that by the Fundamental Theorem of Arithmetic $(m,n) = (r,s)$. Thus different Copy instructions have different code numbers. Similarly we can show that different Jump instructions have different code numbers. Thus $\beta : \mathbf{I} \longrightarrow \mathbb{N}$ is a one–one function.

Clearly $\mathbf{I}$ is not finite. So it follows from Theorem 2.1 that $\mathbf{I}$ is countably infinite. ■

In the process of proving Theorem 2.4 we saw that the function $\beta : \mathbf{I} \longrightarrow \mathbb{N}$ is one–one. Thus each different URM instruction has a different code.

It is, in principle, easy to calculate which number codes a given instruction. Not every natural number codes a URM instruction, but it is also straightforward, in principle, to determine which numbers do code instructions, and, when a number does code an instruction, to determine which instruction it codes. We illustrate this with the following examples and problems.

Example 2.5

(a) $\beta(Z(3)) = 6 \times 3 - 3 = 15$.

(b) $\beta(S(3)) = 6 \times 3 = 18$.

(c) $\beta(C(4,2)) = 2^4\,3^2 + 1 = 145$.

(d) $\beta(J(2,3,6)) = 2^2\,3^3\,5^6 + 2 = 1687\,502$. ♦

Example 2.6

(a) The number 42 is a multiple of 6 and hence it codes a Successor instruction. Indeed as $42 = 6 \times 7$ it codes the instruction $S(7)$.

(b) The number 99 is a multiple of 3 but not a multiple of 6. Hence it codes a Zero instruction. Since $99 + 3 = 102 = 6 \times 17$, it follows that $99 = 6 \times 17 - 3$ and hence 99 codes the instruction $Z(17)$.

(c) The number 649 has remainder 1 when divided by 3. So if it codes a URM instruction, it must code a Copy instruction. Since $649 - 1 = 648 = 2^3\,3^4$, it follows that $649 = 2^3\,3^4 + 1$ and hence 649 codes the instruction $C(3,4)$.

(d) The number 452 has remainder 2 when divided by 3. So if it codes a URM instruction, it must code a Jump instruction. Since $452 - 2 = 450 = 2^1\,3^2\,5^2$, it follows that $452 = 2^1\,3^2\,5^2 + 2$ and hence 452 codes the instruction $J(1,2,2)$.

(e) The number 86 has remainder 2 when divided by 3. So if it codes a URM instruction, it must code a Jump instruction. Since $86 - 2 = 84 = 2^2\,3^1\,7^1$, which is not of the form $2^m\,3^n\,5^q$, it follows that the number 86 does not code a URM instruction.

(f) The number 10 has remainder 1 when divided by 3. So if it codes a URM instruction, it must code a Copy instruction. Now $10 - 1 = 3^2$ which is of the form $2^m\,3^n$ with $m = 0$. But, in the Copy instruction $C(m,n)$, m is the number of a register and so the value $m = 0$ is not allowed. Thus the number 10 does not code a URM instruction. ♦

Problem 2.4

Give the code numbers for the following URM instructions.

(a) $Z(5)$ (b) $S(4)$ (c) $C(5,1)$ (d) $J(3,2,2)$ (e) $J(1,1,1)$

Problem 2.5

Determine which of the following numbers code URM instructions and, in the cases where they do, find the instructions which they code.

(a) 24 (b) 25 (c) 26 (d) 27
(e) 289 (f) 272 (g) 50 000

We are now ready to prove the main result of this subsection. We let **P** be the set of all URM programs.

> ***Theorem 2.5***
>
> The set **P** of URM programs is countably infinite.

Proof

We define a function $\gamma : \mathbf{P} \longrightarrow \mathbb{N}$ as follows. Let P be a URM program and suppose that P consists of k instructions as follows.

γ is the Greek letter 'gamma'.

$$\begin{array}{ll} 1 & I_1 \\ 2 & I_2 \\ \vdots & \vdots \\ k & I_k \end{array}$$

Then we put

$$\gamma(P) = p_1^{\beta(I_1)} p_2^{\beta(I_2)} \dots p_k^{\beta(I_k)}$$

where $p_1, p_2, \dots, p_k$ are the first k prime numbers and $\beta : \mathbf{I} \longrightarrow \mathbb{N}$ is the function defined earlier. From the Fundamental Theorem of Arithmetic and the fact that β is a one–one function (see Theorem 2.4), it follows that γ is a one–one function. Clearly $\mathbf{P}$ is not finite. So it follows from Theorem 2.1 that $\mathbf{P}$ is countably infinite. ■

The function γ will play a very important role later, as its values represent all URM programs.

> ***Definition 2.3***
>
> For each URM program P, $\gamma(P)$ is called the *code number* of P, or *code* of P for short. We say that $\gamma(P)$ *codes* the program P.

In the process of proving Theorem 2.5 we saw that the function $\gamma : \mathbf{P} \longrightarrow \mathbb{N}$ is one–one. Thus each different URM program has a different code, i.e. the number $\gamma(P)$ is unique to P.

Example 2.7

Consider the following program P:

$$\begin{array}{ll} 1 & J(2,3,5) \\ 2 & S(1) \\ 3 & S(3) \\ 4 & J(1,1,1) \end{array}$$

We calculate the code numbers for the instructions in this program as follows:

$$\begin{aligned} &\beta(J(2,3,5)) = 2^2\,3^3\,5^5 + 2 = 337\,502, \\ &\beta(S(1)) = 6 \times 1 = 6, \\ &\beta(S(3)) = 6 \times 3 = 18, \\ &\beta(J(1,1,1)) = 2^1\,3^1\,5^1 + 2 = 32. \end{aligned}$$

Thus

$$\gamma(P) = 2^{337\,502}\,3^6\,5^{18}\,7^{32}.$$

As you can see, although in principle this way of mapping URM programs to natural numbers is straightforward, in practice it leads to very large numbers even for quite short and simple URM programs. For example, the number above has more than 100 000 digits when written out in standard format as a number to base 10. ♦

Problem 2.6

Find $\gamma(P)$ where P is the following URM program.

1 $J(1,2,4)$
2 $S(2)$
3 $C(2,1)$

Problem 2.7

Find the URM program P such that

$$\gamma(P) = 2^{11\,252}\,3^6\,5^{32}.$$

Since, by definition, each URM-computable function of one variable is computed by at least one URM program, we can now deduce that the set of URM-computable functions of one variable is countably infinite, as follows.

Theorem 2.6

The set of URM-computable functions of one variable is countably infinite.

There is a similar result for the set of URM-computable functions of k variables for each $k \geq 2$.

Proof

Let U be the set of URM-computable functions of one variable. For each $f \in U$, let P_f be a URM program which computes f, say for definiteness the program P_f with the smallest code $\gamma(P_f)$. Since the same URM program cannot compute two different functions of one variable, the function $h : U \longrightarrow \mathbb{N}$ defined by $h(f) = \gamma(P_f)$ is one–one. Since U is not finite, it follows from Theorem 2.1 that U is countably infinite. ■

Since each primitive recursive function is URM-computable (Theorem 1.1), it follows from Theorems 2.6 and 2.2 that the set of primitive recursive functions of one variable is countably infinite.

In the next subsection we show that the set of all functions of one variable is *not* countably infinite. It will then follow, from Theorem 2.6, that there exist functions which are not URM-computable.

2.3 Uncountably infinite sets

We have shown that certain infinite sets are countably infinite. It was key to Cantor's ideas that he was able to show that there is more than one kind of infinite set and, in particular, that there are infinite sets which are not countably infinite.

We shall show that the set $\mathbb{R}$ of real numbers is not countably infinite. The proof will involve decimal expansions of real numbers. Each real number has an infinite decimal expansion. In the case of those fractions which are normally regarded as having terminating decimals, such as $\frac{1}{50} = 0.02$, we can turn them into infinite decimal expansions by supposing that all the remaining digits are 0s. Thus

$$\tfrac{1}{50} = 0.02000\ldots.$$

In fact, fractions of this kind have two infinite decimal expansions, since also $\frac{1}{50} = 0.01999\ldots$ with a recurring sequence of 9s. But this is not relevant here. To avoid ambiguity, we assume that in such cases we choose the decimal expansion ending in an infinite sequence of 0s.

Theorem 2.7

The set $\mathbb{R}$ of real numbers is not countably infinite.

Proof

We shall show that there is no one–one correspondence between the set $\mathbb{N}$ of natural numbers and the set $\mathbb{R}$ of real numbers. For suppose that $f : \mathbb{N} \longrightarrow \mathbb{R}$ is a function. We shall show that the function f cannot be onto and thus that f is not a one–one correspondence between $\mathbb{N}$ and $\mathbb{R}$.

Recall that to say that the function f is onto means that for each $x \in \mathbb{R}$ there is at least one $n \in \mathbb{N}$ with $f(n) = x$. So to show that f is not onto, we need to find a number $x \in \mathbb{R}$ which is different from all the numbers in the set $\{f(n) : n \in \mathbb{N}\}$.

Let $d_{0,0}$ be the integer before the decimal point in the decimal expansion of the real number $f(0)$ and let e_0 be an integer different from $d_{0,0}$. For $n, k \geq 1$, let $d_{n,k}$ be the kth digit after the decimal point in the decimal expansion of the real number $f(n)$ and let

$$e_n = \begin{cases} 1, & \text{if } d_{n,n} \neq 1, \\ 2, & \text{if } d_{n,n} = 1. \end{cases}$$

The essential point about these definitions is that $e_n \neq d_{n,n}$ for all $n \in \mathbb{N}$. Now put

$$x = e_0 + \sum_{n=1}^{\infty} \frac{e_n}{10^n}.$$

In the decimal expansion of x, e_0 is the integer before the decimal point and, for $n \geq 1$, e_n is the nth digit after the decimal point: that is,

$$x = e_0 \cdot e_1 e_2 \ldots e_n \ldots .$$

Since $e_0 \neq d_{0,0}$, $x \neq f(0)$. Also, for each $n \in \mathbb{N}$ with $n \geq 1$, the nth digit e_n in the decimal expansion of x is different from the nth digit $d_{n,n}$ in the decimal expansion of $f(n)$ and so $x \neq f(n)$. Thus x is a real number which is not in the set $\{f(n) : n \in \mathbb{N}\}$. Hence f is not an onto function. ■

The argument used in the proof above, due to Cantor, is rightly famous. If we set out the decimal expansions of the numbers $f(0), f(1), \ldots$ in a doubly infinite array, the numbers $d_{0,0}, d_{1,1}, \ldots$ occur on the diagonal of the array.

$$\begin{array}{lccccccc}
f(0) = & \underline{d_{0,0}} \cdot & d_{0,1} & d_{0,2} & d_{0,3} & d_{0,4} & d_{0,5} & \ldots \\
f(1) = & d_{1,0} \cdot & \underline{d_{1,1}} & d_{1,2} & d_{1,3} & d_{1,4} & d_{1,5} & \ldots \\
f(2) = & d_{2,0} \cdot & d_{2,1} & \underline{d_{2,2}} & d_{2,3} & d_{2,4} & d_{2,5} & \ldots \\
f(3) = & d_{3,0} \cdot & d_{3,1} & d_{3,2} & \underline{d_{3,3}} & d_{3,4} & d_{3,5} & \ldots \\
f(4) = & d_{4,0} \cdot & d_{4,1} & d_{4,2} & d_{4,3} & \underline{d_{4,4}} & d_{4,5} & \ldots \\
f(5) = & d_{5,0} \cdot & d_{5,1} & d_{5,2} & d_{5,3} & d_{5,4} & \underline{d_{5,5}} & \ldots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots &
\end{array}$$

We then choose the real number x so that, for each of the numbers $f(0), f(1), \ldots$ in the array, x differs from the nth number in the nth decimal place. Thanks to the form of the argument, it is often referred to as *Cantor's diagonal argument.* It is a powerful method of proof with many significant applications. One of these follows immediately.

> ***Theorem 2.8***
>
> The set of all functions of one variable is infinite but not countably infinite.

Proof

Let F be the set of all functions from $\mathbb{N}$ to $\mathbb{N}$. Clearly F is infinite since it includes for each $k \in \mathbb{N}$ the constant function $C_k : n \longmapsto k$ and these are all different. As in the proof of Theorem 2.7, we use a diagonal argument to show that there is no function from $\mathbb{N}$ to F which is onto.

Suppose $\Phi : \mathbb{N} \longrightarrow F$ is a function and for each $n \in \mathbb{N}$ let $f_n : \mathbb{N} \longrightarrow \mathbb{N}$ be the function $\Phi(n)$. Now define $g : \mathbb{N} \longrightarrow \mathbb{N}$ by the formula

Φ is the Greek letter 'capital phi'.

$$g(n) = f_n(n) + 1.$$

Then $g \in F$ but, for each $n \in \mathbb{N}$, $g(n) \neq f_n(n)$ and hence $g \neq f_n$. Since g is an element of F which is different from all of the values taken by the function Φ, it follows that Φ is not an onto function. So there is no function $\Phi : \mathbb{N} \longrightarrow F$ which is onto and it follows that F is not countably infinite. ■

Two functions are *equal* if they have the same domain and have the same effect on each element of that domain.

Again, if we set out the values of the functions $f_0, f_1, \ldots$ in a doubly infinite array, we see that the values of g are chosen to differ from the corresponding values on the diagonal. It then follows that g is different from each of the functions $f_0, f_1, \ldots$.

	0	1	2	3	4	5	
f_0	$\underline{f_0(0)}$	$f_0(1)$	$f_0(2)$	$f_0(3)$	$f_0(4)$	$f_0(5)$	$\cdots$
f_1	$f_1(0)$	$\underline{f_1(1)}$	$f_1(2)$	$f_1(3)$	$f_1(4)$	$f_1(5)$	$\cdots$
f_2	$f_2(0)$	$f_2(1)$	$\underline{f_2(2)}$	$f_2(3)$	$f_2(4)$	$f_2(5)$	$\cdots$
f_3	$f_3(0)$	$f_3(1)$	$f_3(2)$	$\underline{f_3(3)}$	$f_3(4)$	$f_3(5)$	$\cdots$
f_4	$f_4(0)$	$f_4(1)$	$f_4(2)$	$f_4(3)$	$\underline{f_4(4)}$	$f_4(5)$	$\cdots$
f_5	$f_5(0)$	$f_5(1)$	$f_5(2)$	$f_5(3)$	$f_5(4)$	$\underline{f_5(5)}$	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	

Since the set of URM-computable functions of one variable is countably infinite (Theorem 2.6), it follows from Theorem 2.8 that there exist functions of one variable which are not URM-computable. We can hence deduce, using Theorem 1.1, that there exist functions of one variable which are not primitive recursive.

By themselves these general existence results are neither particularly startling nor informative. However, in the next unit we shall see that the ideas we have used here can be developed to give us more specific information and provide some actual examples of important functions which turn out not to be URM-computable. We shall also see how the ideas here leads us to conclude that there are URM-computable functions which are not primitive recursive.

Before reaching this stage, we return to our investigation of primitive recursive functions. Our aim will be to show that the functions we have used to code URM programs are primitive recursive. This also paves the way for significant results in the next unit.

3 PRIMITIVE RECURSIVE CODING

In this section we are going to return to our study of primitive recursive functions. Our aim will be to show that the functions we have used to code URM programs are primitive recursive. We shall need to use the techniques we have already established for showing that functions are primitive recursive, as well as some new ones.

3.1 Enumerating the primes

The key to our coding is the uniqueness of the prime decomposition of natural numbers greater than 1. You should find it plausible, therefore, that we shall need to know that the function which enumerates the prime numbers is primitive recursive. For the time being we shall use standard function notation and let p be this function. Hence, for $n \geq 1$, $p(n)$will be the nth prime number.

Later we shall return to our earlier notation and write p_n instead of $p(n)$.

First we get a minor point out of the way. We define the function p so that $p(1)$ is the first prime number, that is $p(1) = 2$. This leaves open what we should do about the value of $p(0)$. The choice does not matter greatly. We choose to put $p(0) = 1$. However, we emphasize that 1 is *not* a prime number.

Having specified a value for $p(0)$, the natural way to define p by recursion is to put

$$p(n+1) = \text{the smallest natural number } y \text{ such that } y \text{ is a prime number and } p(n) < y.$$

In order to show that the resulting function is primitive recursive we need to do two things. We need to show that the property of being a prime number is primitive recursive. Recall that by this we mean that the characteristic function of the set of prime numbers is primitive recursive. We have also to find some way of handling functions defined using the operation 'the smallest natural number y such that ...'. We do not have to worry about the inequality in the above definition of $p(n+1)$ as we already know that the relation $<$ is primitive recursive.

We begin by showing that the property of being a prime number is primitive recursive. Let Pr be the set of prime numbers. We shall show that its characteristic function χ_{Pr}, given by

$$\chi_{Pr}(n) = \begin{cases} 1, & \text{if } n \text{ is a prime number,} \\ 0, & \text{otherwise,} \end{cases}$$

is primitive recursive.

A standard way to characterize prime numbers is to say that a natural number n is prime if and only if it has exactly two divisors, 1 and n. Conveniently this excludes 0 which is divisible by every natural number and 1 which has only one divisor. Thus to be able to specify whether or not a number is prime, we need to be able to count its divisors.

Now for $n, y \in \mathbb{N}$ with $y \neq 0$,

$$y \text{ is a divisor of } n \iff \text{rem}(n, y) = 0$$

where rem is the remainder function which we have shown to be primitive recursive in Example 1.10. If we define the function $\text{div} : \mathbb{N}^2 \longrightarrow \mathbb{N}$ by

$$\text{div}(n, y) = \chi_{eq}(\text{rem}(n, y), 0)$$

where χ_{eq} is the characteristic function of the equality relation, we have that for $y \neq 0$

$$\text{div}(n, y) = \begin{cases} 1, & \text{if } y \text{ is a divisor of } n, \\ 0, & \text{otherwise.} \end{cases}$$

From Example 1.8 we know that χ_{eq} is a primitive recursive function. Since div is obtained by substitution from the primitive recursive functions χ_{eq} and rem using constants, it follows that div is primitive recursive.

The function div can be used to count the number of divisors of x in the following way. For $n \neq 0$, any divisor y of n satisfies $y \leq n$. It follows that

$$\sum_{y=1}^{n} \mathrm{div}(n, y) = \text{the number of divisors of } n.$$

Hence a natural number $n > 0$ is prime if and only if $\sum_{y=1}^{n} \mathrm{div}(n, y) = 2$.

Thus, for $n \neq 0$,

$$\chi_{Pr}(n) = \chi_{eq}\left(\sum_{y=1}^{n} \mathrm{div}(n, y), 2\right).$$

To complete the proof that χ_{Pr} is a primitive recursive function, we need to show that the function $n \longmapsto \sum_{y=1}^{n} \mathrm{div}(n, y)$ is primitive recursive. To deal with this we prove a general theorem which has other applications.

Theorem 3.1 Bounded summation

If the function $f : \mathbb{N}^{k+1} \longrightarrow \mathbb{N}$ is primitive recursive, then so also is the function $g : \mathbb{N}^{k+1} \longrightarrow \mathbb{N}$ given by

$$g(n_1, n_2, \ldots, n_k, z) = \begin{cases} 0, & \text{if } z = 0, \\ \sum_{y=1}^{z} f(n_1, n_2, \ldots, n_k, y), & \text{otherwise.} \end{cases}$$

We use the term *bounded sum* for $\sum_{y=1}^{z} a_y$ where z is a positive integer to distinguish it from the infinite sum $\sum_{y=1}^{\infty} a_y$ often used in other parts of mathematics. The z provides the 'bound'.

Proof

The function g satisfies the equations

$$g(n_1, n_2, \ldots, n_k, 0) = 0,$$
$$g(n_1, n_2, \ldots, n_k, z+1) = g(n_1, n_2, \ldots, n_k, z) + f(n_1, n_2, \ldots, n_k, z+1).$$

This shows that g is defined by primitive recursion from the primitive recursive functions add and f using constants. Hence it follows that g is primitive recursive. ■

add is used twice, once to add g and f and once to obtain $z + 1$.

There is an analogous result for products. For a positive integer z, we use the notation $\prod_{y=1}^{z} a_y$ for the product $a_1 \times a_2 \times \cdots \times a_z$.

As z is a positive integer we call this a *bounded product.*

Theorem 3.2 Bounded product

If the function $f : \mathbb{N}^{k+1} \longrightarrow \mathbb{N}$ is primitive recursive, then so also is the function $g : \mathbb{N}^{k+1} \longrightarrow \mathbb{N}$ given by

$$g(n_1, n_2, \ldots, n_k, z) = \begin{cases} 1, & \text{if } z = 0, \\ \prod_{y=1}^{z} f(n_1, n_2, \ldots, n_k, y), & \text{otherwise.} \end{cases}$$

Problem 3.1

Prove Theorem 3.2.

Now we have the following result.

> ***Theorem 3.3***
>
> The characteristic function χ_{Pr} of the set Pr of prime numbers is primitive recursive.

Proof

In the discussion preceding Theorem 3.1 we found that, for $n > 0$,

$$\chi_{Pr}(n) = \chi_{eq}\left(\sum_{y=1}^{n} \mathrm{div}(n, y), 2\right).$$

Now let $g : \mathbb{N}^2 \longrightarrow \mathbb{N}$ be the function given by

$$g(n, z) = \begin{cases} 0, & \text{if } z = 0, \\ \displaystyle\sum_{y=1}^{z} \mathrm{div}(n, y), & \text{otherwise.} \end{cases}$$

Since div is a primitive recursive function, it follows from Theorem 3.1 that g is a primitive recursive function. Also, for $n > 0$,

$$g(n, n) = \sum_{y=1}^{n} \mathrm{div}(n, y).$$

Now let $h : \mathbb{N} \longrightarrow \mathbb{N}$ be the function given by $h(n) = g(n, n)$, which is primitive recursive (by Problem 1.4). We observe that, for all $n \in \mathbb{N}$ (i.e. for $n > 0$ and $n = 0$),

$$\chi_{Pr}(n) = \chi_{eq}(h(n), 2).$$

Remember that $\chi_{Pr}(0) = 0$ and that $g(0, 0) = 0$.

Thus χ_{Pr} is obtained by substitution from the primitive recursive functions χ_{eq} and h using constants, and so is a primitive recursive function. ■

We turn now to the operation 'the smallest natural number y such that ...'. We abbreviate this by μy; here μ is is used to represent 'minimum'. So we can read μy as 'the minimum y such that ...'.

μ is the Greek letter 'mu', pronounced *mew* (as in 'mews').

Recall that in our preliminary work on the function p which enumerates the prime numbers we noted that

$$p(n+1) = \text{the smallest natural number } y \text{ such that } y \text{ is a prime number and } p(n) < y.$$

Now this can be written as

$$p(n+1) = \mu y\,(\chi_{Pr}(y) = 1 \text{ and } p(n) < y). \tag{3.1}$$

Now, as we shall see later (in the proof of Theorem 3.6), the facts that eq, $<$ and χ_{Pr} are primitive recursive enable us to show that the relation '$\chi_{Pr}(y) = 1$ and $m < y$' is primitive recursive. Thus the right-hand side of (3.1) is a function of the form

$$m \longmapsto \mu y\, R(m, y)$$

where R is a primitive recursive relation and we would like to show that this function is primitive recursive. To do this we shall need to introduce the concept of *bounded minimization*, which we shall first apply to functions f before extending it to relations R.

Suppose that $f : \mathbb{N}^{k+1} \longrightarrow \mathbb{N}$ is a function and consider

$$\mu y\,(f(n_1, n_2, \ldots, n_k, y) = 0),$$

that is, for a given $(n_1, n_2, \ldots, n_k) \in \mathbb{N}^k$ we seek the minimum $y \in \mathbb{N}$ such that $f(n_1, n_2 \ldots, n_k, y) = 0$. This is called the *minimization* operation since it picks out the numerically smallest such y. It can also be thought of as a *search* operation as its effect is to search for a solution by checking the values $y = 0$, $y = 1$, $y = 2, \ldots$ in turn.

Of course, there may not be a y such that $f(n_1, n_2, \ldots, n_k, y) = 0$ for certain choices of $(n_1, n_2, \ldots, n_k) \in \mathbb{N}^k$, or indeed for all of them. For example, however $n \in \mathbb{N}$ is chosen, there is no $y \in \mathbb{N}$ such that $n^2 + y^2 + 1 = 0$ and hence $\mu y\,(n^2 + y^2 + 1 = 0)$ is never defined. Similarly $\mu y\,(|n - 2y| = 0)$ is defined when n is even but not when n is odd.

From now on we shall usually write $|n - m|$ rather than $\mathrm{adf}(n, m)$. Recall from Problem 1.6 that the function adf is primitive recursive.

This already gives us a clue that the minimization operation is rather different from substitution and primitive recursion. These last two operations always yield functions, defined for all values of the variables, when they are applied to functions. Minimization, however, as we have just seen, does not. To get around this difficulty, we consider *bounded minimizations*, otherwise known as *bounded searches*. This involves deciding in advance how far to search for solutions, with an arbitrary choice of value if a solution is not found in the given range.

Definition 3.1

Let $f : \mathbb{N}^{k+1} \longrightarrow \mathbb{N}$ be a function and let $(n_1, n_2, \ldots, n_k) \in \mathbb{N}^k$ and $z \in \mathbb{N}$ be fixed. We write the *bounded minimization* operation on f as

$$\mu y \leq z\,(f(n_1, n_2, \ldots, n_k, y) = 0)$$

and specify its effect as follows:

$$\mu y \leq z\,(f(n_1, n_2, \ldots, n_k, y) = 0) = \begin{cases} \text{the smallest } y \in \mathbb{N} \text{ such that} \\ f(n_1, n_2, \ldots, n_k, y) = 0, \\ \text{if there is such a } y \text{ with } y \leq z, \\ z + 1 \text{ otherwise.} \end{cases}$$

The choice of $z + 1$ for the value when there is no solution y less than or equal to z is somewhat arbitrary but none the less very convenient, as we shall see.

This definition means that whatever the choices of $n_1, n_2, \ldots, n_k, z \in \mathbb{N}$, a value for $\mu y \leq z\,(f(n_1, n_2, \ldots, n_k, y) = 0)$ is always defined. Thus when applied to a function f of $k + 1$ variables, bounded minimization always produces another function of $k + 1$ variables, namely the first k variables $n_1, n_2, \ldots, n_k$ that occur in f together with the variable z that occurs in the bound.

Example 3.1

Let the function $g : \mathbb{N}^2 \longrightarrow \mathbb{N}$ be defined by

$$g(n, z) = \mu y \leq z\,(|n - y^2| = 0).$$

Then $g(n, z)$ is equal to the square root of n if n has a square root which is a natural number less than or equal to z, and $z + 1$ otherwise.

For example:

$g(16, 2) = 3$, as 16 does not have a square root which is a natural number less than or equal to 2;

$g(16, 5) = 4$, as the square root of 16 is 4 and $4 \leq 5$;

$g(15, 6) = 7$, as 15 does not have a square root which is a natural number. ♦

Problem 3.2

With g as in Example 3.1, draw up a table of values for $g(n, z)$ for $4 \leq n \leq 9$ and $1 \leq z \leq 6$.

Using the techniques of earlier sections, it is not difficult to see how to give a primitive recursive definition of functions defined by bounded minimization.

Theorem 3.4 Bounded minimization on a function

If $f : \mathbb{N}^{k+1} \longrightarrow \mathbb{N}$ is a primitive recursive function then the function $g : \mathbb{N}^{k+1} \longrightarrow \mathbb{N}$ given by

$$g(n_1, n_2, \ldots, n_k, z) = \mu y \leq z \, (f(n_1, n_2, \ldots, n_k, y) = 0)$$

is also primitive recursive.

Proof

The function g can be defined as follows.

For the sake of brevity, many of the details have been omitted from this proof.

$$\text{(a)} \quad g(n_1, n_2, \ldots, n_k, 0) = \begin{cases} 0, & \text{if } f(n_1, n_2, \ldots, n_k, 0) = 0, \\ 1, & \text{otherwise.} \end{cases}$$

$$\text{(b)} \quad g(n_1, n_2, \ldots, n_k, z+1) = \begin{cases} g(n_1, n_2, \ldots, n_k, z), & \text{if } g(n_1, n_2, \ldots, n_k, z) \leq z, \\ z+1, & \text{if } g(n_1, n_2, \ldots, n_k, z) = z+1 \\ & \text{and } f(n_1, n_2, \ldots, n_k, z+1) = 0, \\ z+2, & \text{otherwise.} \end{cases}$$

The correctness of (a) should be clear. The cases in (a) are clearly mutually exclusive and exhaustive. Furthermore, using the result quoted after Problem 1.15, we can deduce that the relations defining the cases are primitive recursive. Also, since the formulas 0 and 1 for the functions on the right-hand side in (a) are constants, these functions are primitive recursive. Hence, by Theorem 1.5, the function in (a) is primitive recursive.

Let us now check that (b) is correct. Note first that if $g(n_1, n_2, \ldots, n_k, z) \leq z$ then the equation $f(n_1, n_2, \ldots, n_k, y) = 0$ has a solution for some $y \leq z$, and the smallest value of $y \leq z$ which solves this equation is also the smallest value of $y \leq z+1$ which solves this equation. So in this case the value of $g(n_1, n_2, \ldots, n_k, z+1)$ is the same as the value of $g(n_1, n_2, \ldots, n_k, z)$. Otherwise there is no value of $y \leq z$ that solves the equation and the value of $g(n_1, n_2, \ldots, n_k, z+1)$ is $z+1$ if $f(n_1, n_2, \ldots, n_k, z+1) = 0$ and $z+2$ otherwise. Thus (b) is correct. Also, this discussion shows that the cases in (b) are mutually exclusive and exhaustive.

Furthermore, using the result quoted after Problem 1.15, we can deduce that the relations defining the cases in (b) are primitive recursive. Also, since the formulas $g(n_1, n_2, \ldots, n_k, z)$, $z+1$ and $z+2$ for the functions on the right-hand side are expressed in terms of $g(n_1, n_2, \ldots, n_k, z)$, the variable z and the primitive recursive function add using constants, these functions are primitive recursive. Hence, by Theorem 1.5, the function in (b) is primitive recursive.

Therefore g is defined by primitive recursion from functions known to be primitive recursive. It follows that g is a primitive recursive function. ■

Example 3.2

Let the function $h : \mathbb{N} \longrightarrow \mathbb{N}$ be defined by

$$h(n) = \begin{cases} \sqrt{n}, & \text{if } n \text{ has a square root} \\ & \text{which is a natural number,} \\ n+1, & \text{otherwise.} \end{cases}$$

Note that $h(0) = 0$, $h(1) = 1$, $h(2) = 3$, $h(3) = 4$, $h(4) = 2$ and so on.

We can use Theorem 3.4 to show that h is primitive recursive.

We do this by writing h in terms of the function g from Example 3.1, which is defined by

$$g(n, z) = \mu y \leq z\,(|n - y^2| = 0).$$

Since $\sqrt{n} \leq n$ for all $n \in \mathbb{N}$, we can be sure that the square root y of n is encountered for some $y \leq n$. Thus

$$h(n) = g(n, n).$$

Therefore, by Problem 1.4, h is primitive recursive if g is.

The function $f : \mathbb{N}^2 \longrightarrow \mathbb{N}$ given by

$$f(n, y) = |n - y^2|$$

is primitive recursive since it is obtained by substitution from the primitive recursive functions adf and mult. Hence, by Theorem 3.4, g is primitive recursive, as required. ♦

In order to use bounded minimization to show that the function p which enumerates the prime numbers is primitive recursive, we need to extend the idea to relations.

Definition 3.2

Let $R(n_1, n_2, \ldots, n_k, y)$ be a $(k+1)$-place relation on $\mathbb{N}^{k+1}$ and let $(n_1, n_2, \ldots, n_k) \in \mathbb{N}^k$ and $z \in \mathbb{N}$ be fixed. We write the *bounded minimization* operation on R as

$$\mu y \leq z\, R(n_1, n_2, \ldots, n_k, y)$$

and specify its effect as follows:

$$\mu y \leq z\, R(n_1, n_2, \ldots, n_k, y) = \begin{cases} \text{the smallest } y \in \mathbb{N} \text{ for which} \\ R(n_1, n_2, \ldots, n_k, y) \text{ holds,} \\ \text{if there is such a } y \text{ with } y \leq z, \\ z + 1 \text{ otherwise.} \end{cases}$$

We can think of Definition 3.1 as a special case of this definition for the relation $f(n_1, n_2, \ldots, n_k, y) = 0$.

We also need to extend Theorem 3.4 to cover relations.

Theorem 3.5 Bounded minimization on a relation

If R is a primitive recursive $(k+1)$-place relation then the function $g : \mathbb{N}^{k+1} \longrightarrow \mathbb{N}$ given by

$$g(n_1, n_2, \ldots, n_k, z) = \mu y \leq z\, R(n_1, n_2, \ldots, n_k, y)$$

is a primitive recursive function.

Problem 3.3

Prove Theorem 3.5.

We use Theorem 3.5 in the proof that the function p which enumerates the prime numbers is primitive recursive. For this application, we need a bound on the size of the nth prime number. In Number Theory, *Unit 2*, it was shown that the nth prime number p_n satisfies the inequality $p_n \leq 2^{2^{n-1}}$. This is a crude bound but it serves our purpose.

Theorem 3.6

The function p which enumerates the prime numbers is primitive recursive.

Proof

From our preliminary work, we have seen that

$$p(n+1) = \mu y\,(\chi_{Pr}(y) = 1 \text{ and } p(n) < y).$$

Consider the relation S given by

$$S(m, y) \iff \chi_{Pr}(y) = 1.$$

We need to make S a two-place rather than a one-place relation so that we can use one of the results in Problem 1.10 later in the proof.

We have

$$\chi_S(m, y) = \chi_{eq}(\chi_{Pr}(y), 1).$$

Therefore χ_S is primitive recursive since it is obtained by substitution from the primitive recursive functions χ_{eq} and χ_{Pr} using constants. Thus the relation S is primitive recursive.

Also the relation $<$ is primitive recursive (by Problem 1.9).

Now define the relation R by

$$R(m, y) \iff S(m, y) \text{ and } m < y \iff \chi_{Pr}(y) = 1 \text{ and } m < y.$$

This is primitive recursive, by Problem 1.10, since S and $<$ are.

Therefore, by Theorem 3.5, the function $g : \mathbb{N}^2 \longrightarrow \mathbb{N}$ given by

$$g(m, z) = \mu y \leq z\,(\chi_{Pr}(y) = 1 \text{ and } m < y)$$

is primitive recursive. We observe that $g(p(n), z) = p(n+1)$ provided that $p(n+1) \leq z$.

Next, let $h : \mathbb{N} \longrightarrow \mathbb{N}$ be defined by $h(n) = \exp(2, \exp(2, n))$. Then h is a primitive recursive function since it is obtained by substitution from the primitive recursive function exp using constants. Also, using the remark preceding the statement of this theorem, we have $p(n+1) \leq 2^{2^n} = h(n)$.

It follows that

$$p(n+1) = g(p(n), h(n)),$$

where g and h are primitive recursive. Thus, using the definition of p given at the start of this subsection, we have

$$p(0) = 1,$$

$$p(n+1) = g(p(n), h(n)),$$

so that p is defined by primitive recursion from the constant 1 and the primitive recursive functions g and h. Thus p is a primitive recursive function. ■

Note that in the proof above we found a bound $h(n)$ on the values of y used to search for the value of $p(n+1)$ in terms of a primitive recursive function h, so that

$$p(n+1) = g(p(n), h(n)).$$

The particular bound used here was a function of n, but it could just as usefully have involved the value of $p(n)$ as well as that of n. For instance, we know from the proof of Euclid's Theorem in Number Theory, *Unit 2*, that the nth prime number p_n satisfies the inequality $p_n \leq p_1 p_2 \dots p_{n-1} + 1$. Hence it also satisfies the inequality $p_n \leq p_{n-1}^{n-1} + 1$. Therefore, using the primitive recursive function exp, we could alternatively have defined $p(n+1)$ by

$$p(n+1) = g(p(n),\ \exp(p(n), n) + 1)$$

to show that p was primitive recursive.

Problem 3.4

Let $f : \mathbb{N} \longrightarrow \mathbb{N}$ be the function defined by

$$f(n) = \mu y\,(n < 3^y).$$

Show that f is primitive recursive. *Hint:* Note that $n < 3^n$ for all $n \in \mathbb{N}$.

Note that $f(0) = 0$, $f(1) = f(2) = 1$, $f(3) = 2$ and so on.

Problem 3.5

Let $f : \mathbb{N} \longrightarrow \mathbb{N}$ be the function defined by

$$f(0) = 1,$$
$$f(n+1) = \mu y\,(f(n) < y \text{ and } y \text{ is not divisible by } 4).$$

Show that f is primitive recursive. *Hint:* You will have to find a bound z expressible as a primitive recursive function h of n and $f(n)$ such that a search on $y \leq z$ will always find $f(n+1)$.

Note that $f(0) = 1$, $f(1) = 2$, $f(2) = 3$, $f(3) = 5$ and so on.

3.2 Coding sequences

Recall that if we have a URM program with instructions $I_1, I_2, \dots, I_k$ then we use the number

$$p_1^{\beta(I_1)} p_2^{\beta(I_2)} \dots p_k^{\beta(I_k)} \tag{3.2}$$

to code the program. Moreover the exponents $\beta(I_1), \beta(I_2), \dots, \beta(I_k)$ are all positive integers. Remember that *every number which codes a URM program is of form (3.2) but not every number of this form codes a URM program.*

Recall that $p_1, p_2, \dots, p_k$ are the first k primes.

In the next subsection we shall show that the set of code numbers of URM programs is a primitive recursive set. In this subsection we shall prove some preliminary results, including showing that the set of all numbers of the form

$$p_1^{a_1} p_2^{a_2} \dots p_k^{a_k},$$

where $a_1, a_2, \dots, a_k$ are positive integers, is primitive recursive. If you are short of time, do not spend too long on these two subsections. You should, however, aim at least to take note of the sets and functions being defined and to try to obtain some appreciation of how they are being used.

Some of the sets and functions defined in Subsections 3.2 and 3.3 will be used again in *Unit 3*.

Let us say that a positive integer n *codes* a finite sequence of *positive* integers $a_1, a_2, \dots, a_k$ if

$$n = p_1^{a_1} p_2^{a_2} \dots p_k^{a_k},$$

and let *Seq* denote the set of code numbers of finite sequences of positive integers. We observe that $n \in Seq$ if and only if n is divisible by all the primes $p_1, p_2, \dots, p_k$ where p_k is the largest prime which divides n.

Clearly the number of distinct prime divisors of a given positive integer will be significant in our study of *Seq*. We call the number of distinct prime divisors of an integer $n \geq 2$ the *length* of n. Let us define $\text{len}: \mathbb{N} \longrightarrow \mathbb{N}$ by putting

$$\text{len}(n) = \begin{cases} \text{the number of distinct} \\ \text{prime divisors of } n, & \text{if } n \geq 2, \\ 0, & \text{otherwise.} \end{cases}$$

So, if n codes a sequence of k positive integers then $\text{len}(n) = k$.

Problem 3.6

Evaluate len(45), len(700) and len(2100).

Now we shall show that len is a primitive recursive function. Note that, since $y < p_y$, it follows that if p_y is a divisor of n then $y < n$. Thus we see that $\text{len}(0) = 0$ and for $n > 0$

$$\text{len}(n) = \sum_{y=1}^{n} \text{div}(n, p_y).$$

Notice that this gives the correct value for $n = 1$.

Remember that $p_y = p(y)$.

We saw in Subsection 3.1 that the functions div and p are primitive recursive, and hence the function $(n, y) \longmapsto \text{div}(n, p_y)$ obtained by substitution from the primitive recursive functions div and p is primitive recursive. Therefore, applying Theorem 3.1 to the function $(n, y) \longmapsto \text{div}(n, p_y)$ we see that the function $g: \mathbb{N}^2 \longrightarrow \mathbb{N}$ given by

$$g(n, z) = \begin{cases} 0, & \text{if } z = 0, \\ \displaystyle\sum_{y=1}^{z} \text{div}(n, p_y), & \text{if } z \neq 0, \end{cases}$$

is primitive recursive. Since

$$\text{len}(n) = g(n, n),$$

we see that len is a primitive recursive function (by Problem 1.4).

Now let χ_{Seq} be the characteristic function of *Seq*. Thus $\chi_{Seq}(n) = 1$ if n codes a finite sequence of positive integers and $\chi_{Seq}(n) = 0$ otherwise. We need to distinguish between numbers such as $2520 = 2^3\,3^2\,5^1\,7^1$ which is divisible by all the prime numbers from 2 up to the largest prime number which divides it (so $\chi_{Seq}(2520) = 1$) and numbers such as $7000 = 2^3\,5^3\,7^1$ which is divisible by the primes 2, 5 and 7 but not by the prime 3 (so $\chi_{Seq}(7000) = 0$).

Problem 3.7

Evaluate $\chi_{Seq}(n)$ for the following values of n.

(a) 30 (b) 60 (c) 42 (d) 200 200

Theorem 3.7

The set *Seq* of code numbers of finite sequences of positive integers is primitive recursive.

Proof

We must show that the characteristic function χ_{Seq} is primitive recursive. We observe that $\chi_{Seq}(n) = 1$ if and only if p_y is a divisor of n for $1 \leq y \leq \text{len}(n)$; that is, if and only if $\text{div}(n, p_y) = 1$ for $1 \leq y \leq \text{len}(n)$. The numbers $\text{div}(n, p_y)$ for $1 \leq y \leq \text{len}(n)$ are all equal to 1 if and only if their product is 1. Thus we can define χ_{Seq} by

$$\chi_{Seq}(n) = \begin{cases} \prod_{y=1}^{\text{len}(n)} \text{div}(n, p_y), & \text{if } n > 1, \\ 0, & \text{otherwise.} \end{cases}$$

Applying the result of Theorem 3.2 to the primitive recursive function $(n, y) \longmapsto \text{div}(n, p_y)$ we see that the function $g : \mathbb{N}^2 \longrightarrow \mathbb{N}$ given by

$$g(n, z) = \begin{cases} 1, & \text{if } z = 0, \\ \prod_{y=1}^{z} \text{div}(n, p_y), & \text{if } z \neq 0, \end{cases}$$

is primitive recursive. We have

$$\chi_{Seq}(n) = \begin{cases} g(n, \text{len}(n)), & \text{if } n > 1, \\ 0, & \text{otherwise.} \end{cases}$$

The formula $g(n, \text{len}(n))$ for the first function on the right-hand side is obtained by substitution from the primitive recursive functions g and len, and so that function is primitive recursive. Since the relation $>$ is primitive recursive, using the remark after Problem 1.15 we can deduce that the relations defining the cases are primitive recursive. Moreover, these relations are clearly mutually exclusive and exhaustive. So, by Theorem 1.5, it follows that χ_{Seq} is primitive recursive. ■

Suppose that a number n codes a finite sequence $a_1, a_2, \ldots, a_k$ of positive integers so that

$$n = p_1^{a_1} p_2^{a_2} \ldots p_k^{a_k}.$$

Then n codes a URM program if and only if the exponents $a_1, a_2, \ldots, a_k$ are the code numbers of URM instructions. So decoding the code of a URM program will involve extracting from a given number the exponents of the prime numbers in its prime decomposition. We show next that this can be done by a primitive recursive function. In line with the tradition of the subject, we use the notation $(n)_j$ for the exponent of the jth prime number p_j in the prime decomposition of the integer n where $n \geq 2$. For a prime p_j that does not divide n, so does not appear in its prime decomposition, we can regard its exponent $(n)_j$ in this decomposition as being 0. Since we want $(n, j) \longmapsto (n)_j$ to be a function with domain $\mathbb{N}^2$, we choose to put $(n)_0 = 0$ and $(0)_j = (1)_j = 0$ for all j.

Example 3.3

As $2520 = 2^3\, 3^2\, 5^1\, 7^1$ and 2, 3, 5 and 7 are respectively the 1st, 2nd, 3rd and 4th prime numbers we see that

$$(2520)_1 = 3, \quad (2520)_2 = 2, \quad (2520)_3 = 1, \quad (2520)_4 = 1.$$

Also $(2520)_0 = 0$, by definition. Finally, for $j > 4$, p_j does not appear in the prime decomposition of 2520, so that $(2520)_j = 0$ for $j > 4$. ♦

Problem 3.8

Find the values of

$$(360)_j, \quad (243)_j \quad \text{and} \quad (462)_j$$

for all natural numbers j.

For $n \neq 0$ and $j \neq 0$, $(n)_j$ is the largest value of k for which $p(j)^k$ is a divisor of n, where p is the primitive recursive function which enumerates the primes. It seems at first that we are going to need an operator 'the largest k such that ...'. Fortunately this not necessary as the largest k such that $p(j)^k$ is a divisor of n is the same as the smallest k such that $p(j)^{k+1}$ is *not* a divisor of n. So our proof that $(n, j) \longmapsto (n)_j$ is primitive recursive will involve bounded minimization and we need a convenient bound. We observe that if $r \geq n$ and $j \neq 0$, then $p(j)^r \geq 2^r \geq 2^n > n$. Thus the exponent of a prime number $p(j)$ in the prime decomposition of n must be less than n. Hence for $n \neq 0$ and $j \neq 0$, $(n)_j$ is the smallest natural number k such that $k \leq n$ and $p(j)^{k+1}$ is not a divisor of n.

Theorem 3.8

The function $(n, j) \longmapsto (n)_j$ is primitive recursive.

Proof

The condition that $p(j)^{k+1}$ is not a divisor of n can be expressed by the relation

$$\operatorname{div}(n, p(j)^{k+1}) = 0.$$

The characteristic function of this relation is primitive recursive since it is obtained by substitution from the primitive recursive functions χ_{eq}, div, p, exp and add using constants. Hence the relation

$$R(n, j, k) \iff \operatorname{div}(n, p(j)^{k+1}) = 0$$

is primitive recursive. From the discussion above, we can see that

$$(n)_j = \begin{cases} \mu k \leq n\, R(n, j, k), & \text{if } n \neq 0 \text{ and } j \neq 0, \\ 0, & \text{otherwise.} \end{cases}$$

Now, since $R(n, j, k)$ is primitive recursive, the function given by $\mu k \leq z\, R(n, j, k)$ is primitive recursive, by Theorem 3.5. Hence, the function given by $\mu k \leq n\, R(n, j, k)$ is primitive recursive, by the remark following Problem 1.4. Since the relation eq is primitive recursive, using Problem 1.10 and the remark after Problem 1.15 we can deduce that the relations defining the cases are primitive recursive. Moreover, these relations are clearly mutually exclusive and exhaustive. So, by Theorem 1.5, it follows that the function $(n, j) \longmapsto (n)_j$ is primitive recursive. ■

We have now assembled the machinery needed to show that the functions involved in our coding of URM programs are all primitive recursive. We turn to this in the final subsection.

3.3 Codes of URM programs

In this subsection we show that the sets of code numbers of URM instructions and URM programs are primitive recursive. Remember that, if you are short of time, you should not spend too much time on this subsection; try simply to obtain an appreciation of what is involved.

We begin by considering the URM instructions. We let *Zinstr*, *Sinstr*, *Cinstr* and *Jinstr* be the sets of code numbers of the Zero, Successor, Copy and Jump instructions respectively and let *Instr* be the union of these sets; that is, *Instr* is the set of code numbers of URM instructions. We shall show that *Instr* is a primitive recursive set.

Recall from the previous subsection that a number k codes a sequence of s positive integers if and only if $\chi_{Seq}(k) = 1$ and $\operatorname{len}(k) = s$.

Theorem 3.9

The set *Instr* of code numbers of URM instructions is primitive recursive.

Proof

Since the union of a finite number of primitive recursive sets is primitive recursive (by the remark after Problem 1.8), it is enough to show that the sets *Zinstr*, *Sinstr*, *Cinstr* and *Jinstr* are primitive recursive.

First consider

$$Zinstr = \{\beta(Z(n)) : n \in \mathbb{Z}^+\} = \{6n - 3 : n \in \mathbb{Z}^+\}.$$

Thus *Zinstr* is the set of natural numbers which are divisible by 3 but not by 6. So the characteristic function of this set is given by

$$\chi_{Zinstr}(k) = \mathrm{div}(k, 3)\,\overline{\mathrm{sg}}(\mathrm{div}(k, 6))$$

and this is a primitive recursive function since it is obtained by substitution from the primitive recursive functions mult, div and $\overline{\mathrm{sg}}$ using constants. Hence *Zinstr* is a primitive recursive set.

Next consider

$$Sinstr = \{\beta(S(n)) : n \in \mathbb{Z}^+\} = \{6n : n \in \mathbb{Z}^+\}.$$

Its characteristic function is given by

$$\chi_{Sinstr}(k) = \mathrm{sg}(k)\,\mathrm{div}(k, 6)$$

and this is primitive recursive function since it is obtained by substitution from the primitive recursive functions mult, sg and div using constants. Hence *Sinstr* is a primitive recursive set.

Recall that $\beta(C(m, n)) = 2^m\,3^n + 1$. Hence $k \in Cinstr$ if and only if the remainder when k is divided by 3 is 1 and $k - 1$ codes a pair of positive integers. Thus

$$k \in Cinstr \iff \mathrm{rem}(k, 3) = 1 \text{ and } \chi_{Seq}(k \dot{-} 1) = 1 \text{ and } \mathrm{len}(k \dot{-} 1) = 2.$$

If we introduce the properties $P(k) \iff \mathrm{rem}(k, 3) = 1$ and $Q(k) \iff \mathrm{len}(k \dot{-} 1) = 2$, we have

Remember that a property is a one-place relation.

$$\chi_{Cinstr}(k) = \chi_P(k)\chi_{Seq}(k \dot{-} 1)\chi_Q(k).$$

The function χ_P is primitive recursive since it is obtained by substitution from the primitive recursive functions χ_{eq} and rem using using constants. The function χ_Q is primitive recursive since it is obtained by substitution from the primitive recursive functions χ_{eq}, len and $\dot{-}$ using constants. Therefore the function χ_{Cinstr} is primitive recursive since it is obtained by substitution from the primitive recursive functions mult, χ_P, χ_{Seq}, $\dot{-}$ and χ_Q using constants. Hence *Cinstr* is a primitive recursive set.

Finally, since $\beta(J(m, n, q)) = 2^m\,3^n\,5^q + 2$, we see that

$$k \in Jinstr \iff \mathrm{rem}(k, 3) = 2 \text{ and } \chi_{Seq}(k \dot{-} 2) = 1 \text{ and } \mathrm{len}(k \dot{-} 2) = 3.$$

If we introduce the properties $R(k) \iff \mathrm{rem}(k, 3) = 2$ and $S(k) \iff \mathrm{len}(k \dot{-} 1) = 3$, we have

$$\chi_{Jinstr}(k) = \chi_R(k)\chi_{Seq}(k \dot{-} 2)\chi_S(k),$$

and in a similar manner to above we can deduce that χ_{Jinstr} is a primitive recursive function. Hence *Jinstr* is a primitive recursive set. ■

We are now ready to show that the set of code numbers of URM programs is primitive recursive. We let *Prog* be this set. Thus

$$Prog = \{\gamma(P) : P \in \mathbf{P}\}.$$

> ***Theorem 3.10***
>
> The set *Prog* of code numbers of URM programs is primitive recursive.

Proof

A natural number n codes a URM program if and only if it codes a sequence of positive integers which are the code numbers of URM instructions. If n codes such a sequence, $\text{len}(n)$ is the number of terms in the sequence. Also, for $1 \leq j \leq \text{len}(n)$, the number $(n)_j$ is the exponent of the jth prime in the prime decomposition of n. So $(n)_j$ is the jth number in the sequence coded by n. Thus

$$n \in Prog \iff \chi_{Seq}(n) = 1 \text{ and } (n)_j \in Instr \text{ for } 1 \leq j \leq \text{len}(n).$$

Now $(n)_j \in Instr$ if and only if $\chi_{Instr}((n)_j) = 1$. So $(n)_j \in Instr$ for $1 \leq j \leq \text{len}(n)$ if and only if $\chi_{Instr}((n)_j) = 1$ for $1 \leq j \leq \text{len}(n)$, and this is the case if and only if

$$\prod_{j=1}^{\text{len}(n)} \chi_{Instr}((n)_j) = 1.$$

Thus $n \in Prog$ if and only if

$$\chi_{Seq}(n) \times \prod_{j=1}^{\text{len}(n)} \chi_{Instr}((n)_j) = 1.$$

Let us now define the function $g : \mathbb{N}^2 \longrightarrow \mathbb{N}$ by

$$g(n,z) = \begin{cases} 1, & \text{if } z = 0, \\ \displaystyle\prod_{j=1}^{z} \chi_{Instr}((n)_j), & \text{otherwise.} \end{cases}$$

We use this function to obtain the characteristic function of the set *Prog*, which is given by

$$\chi_{Prog}(n) = \chi_{Seq}(n) g(n, \text{len}(n)).$$

We need to define g to ensure that χ_{Prog} is defined if $\text{len}(n) = 0$, i.e. if $n = 0$ or $n = 1$.

Now the function $(n,j) \longmapsto \chi_{Instr}((n)_j)$ is primitive recursive since it is obtained by substitution from the primitive recursive functions χ_{Instr} and $(n,j) \longmapsto (n)_j$. Hence the function g is primitive recursive, by Theorem 3.2. Therefore the function χ_{Prog} is primitive recursive since it is obtained by substitution from the primitive recursive functions mult, χ_{Seq}, g and len. Hence *Prog* is a primitive recursive set. ■

The advantage we gain from knowing that URM instructions and URM programs can be coded in a primitive recursive way will become apparent in the next unit.

SUMMARY

In this unit, primitive recursive functions were introduced. These are the functions which can be obtained from certain basic functions by the operations of substitution and primitive recursion. It follows from work in *Unit 1* that every primitive recursive function is URM-computable. We found that many standard functions such as addition, multiplication and exponentiation are primitive recursive, and we also found some more unexpected examples such as the function which enumerates the prime numbers. Primitive recursiveness for sets of natural numbers and relations between numbers was defined using their characteristic functions. This led us to the important technique of definition by cases for establishing primitive recursiveness.

The 'size' of infinite sets was considered. We defined a set to be countably infinite if it is in one–one correspondence with the set $\mathbb{N}$ of natural numbers. We found that infinite subsets of $\mathbb{N}$ are countably infinite and that $\mathbb{N}^k$, $k > 1$, is countably infinite. We also found that the sets of URM instructions, URM programs and URM functions of one variable are countably infinite. However, the set of all functions of one variable is not countably infinite. So we have found that there are functions which are not URM-computable, and hence which are not primitive recursive.

A standard way of showing that an infinite set X is countably infinite is to show that there is a one–one function from X into $\mathbb{N}$. The values of this function can then be regarded as code numbers for the elements of X. In this way we coded the URM instructions and URM programs. Finally we established that the sets of code numbers of URM instructions and URM programs are primitive recursive. The proofs involved all the techniques introduced earlier in the unit and the important new ones of bounded summation, bounded product and bounded minimization.

A question left open in this unit is whether or not every URM-computable function is primitive recursive. This will be answered negatively in *Unit 3*. Then one more operation will be introduced and we shall find that all URM-computable functions can be obtained from the basic functions using this operation together with substitution and primitive recursion.

OBJECTIVES

We list those topics on which we may set assessment questions to test your understanding of this unit.

After working through the unit you should be able to:

(a) give a proof that a given function is primitive recursive by showing how it can be obtained from functions already known to be primitive recursive using substitutions and primitive recursions;

(b) show that a specified set or relation is primitive recursive by proving that its characteristic function is primitive recursive;

(c) evaluate the functions used to code ordered k-tuples in the proof that $\mathbb{N}^k$, $k > 1$, is countably infinite and, given a natural number, determine which ordered k-tuple, if any, it codes;

(d) prove in simple cases that specified sets are countably infinite;

(e) find the code numbers of specified URM instructions and URM programs;

(f) given the prime decomposition of a number, determine whether or not it codes a URM program and, if so, which program it codes;

(g) determine particular values of functions defined by bounded minimization;

(h) show that a specified function or relation is primitive recursive using techniques introduced in the unit, such as the use of constants, definition by cases, bounded minimization, bounded summation or bounded product applied to a function or relation already known to be primitive recursive;

(i) evaluate particular cases of the functions len, χ_{Seq} and $(n, j) \longmapsto (n)_j$.

ADDITIONAL EXERCISES

Most of these exercises provide further practice, should you feel you need it, in handling the main ideas in the unit on which you are likely to be assessed.

There are a few harder problems, labelled as such in the margin. These are harder than any of the problems you are likely to encounter in the assessment and are included solely as challenges for the interested student.

Section 1

1 Show that, for $k \geq 1$, every constant function of k variables is primitive recursive.

2 Show that, for each $k \in \mathbb{N}$, the sets

$$\{n \in \mathbb{N} : n \neq k\}, \quad \{n \in \mathbb{N} : n > k\}$$

are primitive recursive.

3 The *symmetric difference* of the sets A and B, written $A \Delta B$, is defined by

$$A \Delta B = \{x : (x \in A \text{ and } x \notin B) \text{ or } (x \in B \text{ and } x \notin A)\}.$$

Prove that if A and B are primitive recursive sets then so also is $A \Delta B$.

4 Show that the function f defined by

$$f(n_1, n_2, n_3) = \begin{cases} 2n_2, & \text{if } n_1 + 12 \leq \min(n_2, n_3), \\ n_1^{n_3}, & \text{if } 2n_1 + 3n_2 + 4n_3 = 80, \\ 5, & \text{otherwise,} \end{cases}$$

is primitive recursive.

Section 2

1 Let f_3 be the function used for coding ordered triples of natural numbers in Example 2.3.

(a) Evaluate $f_3(4, 2, 4)$ and $f_3(4, 3, 2)$.

(b) For each of the following values of k determine whether there are natural numbers n_1, n_2, n_3 such that $f_3(n_1, n_2, n_3) = k$ and, if so, determine the values of n_1, n_2 and n_3.

(i) $k = 100$ (ii) $k = 480$ (iii) $k = 490$

2 (a) Prove that if for each $k \in \mathbb{N}$ the set A_k is countably infinite, then the set $\bigcup_{k \in \mathbb{N}} A_k$ is also countably infinite. *Hint:* Consider how you might use one–one functions from A_k to $\mathbb{N}$ for each k together with the Fundamental Theorem of Arithmetic to help you define a suitable one–one function from $\bigcup_{k \in \mathbb{N}} A_k$ to $\mathbb{N}$.

(b) A positive real number x is said to be a *root number* if, for some $k \in \mathbb{N}$ with $k \geq 2$, we have $x^k \in \mathbb{N}$. Prove that the set of root numbers is countably infinite.

Harder problem

The notation $\bigcup_{k \in \mathbb{N}} A_k$ means $A_0 \cup A_1 \cup A_2 \cup \ldots$.

3 (a) Prove that the set $\mathbb{Q}^+$ of positive rational numbers is countably infinite. *Hint:* Every positive rational number can be written in a *unique* way as a/b where a and b are relatively prime positive integers. Exploit this fact to define a one–one function from $\mathbb{Q}^+$ to $\mathbb{N}$.

(b) Show that the set $\mathbb{Q}$ of rational numbers is countably infinite.

Recall, from Number Theory, *Unit 1*, that the rational numbers are the numbers a/b with $a \in \mathbb{Z}$ and $b \in \mathbb{Z}^+$ and that a and b are relatively prime if $\gcd(a, b) = 1$, i.e. if the greatest common divisor of a and b is 1.

4 Prove that the set $\mathscr{P}(\mathbb{N})$ of all subsets of the set $\mathbb{N}$ of natural numbers is infinite but not countably infinite. *Hint:* Use a diagonal argument to show that there is no function from $\mathbb{N}$ to $\mathscr{P}(\mathbb{N})$ which is onto.

Harder problem

This is a special case of what is called *Cantor's Theorem.*

5 (a) Calculate the code number for each of the following URM instructions.

(i) $Z(7)$ (ii) $S(8)$ (iii) $C(5, 2)$ (iv) $J(1, 5, 3)$

(b) In each of the following cases determine whether the given number codes a URM instruction and, if so, determine the URM instruction that it codes.

(i) 72 (ii) 73 (iii) 74 (iv) 75 (v) 152

6 Which is the smallest number that codes a URM instruction?

Section 3

1 Show that the set

$$\{n \in \mathbb{N} : n \text{ is not a multiple of } 3\}$$

is primitive recursive.

2 For any integer $n \geq 1$, $\sigma(n)$ is defined to be the sum of the distinct divisors of n including 1 and n. We put $\sigma(0) = 0$. Prove that $\sigma : \mathbb{N} \longrightarrow \mathbb{N}$ is a primitive recursive function.

This function appears in Number Theory, *Unit 5.*

3 The function $g : \mathbb{N}^2 \longrightarrow \mathbb{N}$ is defined by

$$g(n, z) = \mu y \leq z\,(y^3 = n).$$

Evaluate:

(a) $g(64, 2)$; (b) $g(125, 10)$; (c) $g(25, 10)$.

4 Let $f : \mathbb{N} \longrightarrow \mathbb{N}$ be the function defined by

$$f(n) = \mu y\,(n \leq y \text{ and } y \text{ is a multiple of } 5).$$

Show that f is primitive recursive.

Note that $f(0) = 0$, $f(1) = f(2) = \ldots = f(5) = 5$, $f(6) = 10$ and so on.

5 (a) Evaluate:

(i) $\text{len}(60)$; (ii) $\text{len}(20!)$.

(b) Evaluate:

(i) $\chi_{Seq}(420)$; (ii) $\chi_{Seq}(4900)$; (iii) $\chi_{Seq}(20!)$.

(c) Evaluate:

(i) $(720)_2$; (i) $(7007)_4$; (ii) $(3275\,346\,933)_1$.

6 The *Fibonacci sequence* (Number Theory, *Unit 2*) is the sequence of numbers defined, for $n \in \mathbb{N}$, by

Harder problem

$$F(0) = 0, \quad F(1) = 1 \quad \text{and} \quad F(n+2) = F(n+1) + F(n).$$

This defines F by a type of recursion but, on the face of it, it is not a primitive recursion because instead of defining $F(n+1)$ in terms of $F(n)$, the recursion equation defines $F(n+2)$ in terms of $F(n+1)$ and $F(n)$. The aim of this exercise is to show that, none the less, F is a primitive recursive function.

We let G be the function defined by

$$G(n) = 2^{F(n)}3^{F(n+1)}.$$

(a) Prove that G is a primitive recursive function.

(b) Deduce that F is a primitive recursive function.

7 Show that the function f defined by

Harder problem

$$f(n) = \begin{cases} \text{the largest prime number} \\ \text{which is a divisor of } n, & \text{if } n \geq 2, \\ n, & \text{otherwise,} \end{cases}$$

is primitive recursive.

SOLUTIONS TO THE PROBLEMS

Solution 1.1

The function C_0 is the basic primitive recursive function zero.

Now suppose that, for some natural number m, C_m is a primitive recursive function. Since $\text{succ}(m) = m + 1$, we see that

$$C_{m+1}(n) = \text{succ}(C_m(n)).$$

Since C_m is a primitive recursive function and C_{m+1} is obtained by substitution from the basic primitive recursive function succ and C_m, it follows that C_{m+1} is a primitive recursive function.

We are here making use of the fact noted immediately after Definition 1.2.

It now follows from the generalized Principle of Mathematical Induction (Number Theory, *Unit 1*) that C_a is a primitive recursive function for all natural numbers a.

Solution 1.2

We observe that

$$an + b = \text{add}(\text{mult}(a, n), b) = \text{add}(\text{mult}(a, n), C_b(n))$$

where C_b is a constant function as defined in Problem 1.1. Thus the function given by $n \longmapsto an + b$ is obtained by substitution from the functions add, C_b and h, where $h : \mathbb{N} \longrightarrow \mathbb{N}$ is given by

$$h(n) = \text{mult}(a, n).$$

Notice that, since a is fixed, $\text{mult}(a, n)$ is a function of just one variable.

We know that add and C_b are primitive recursive (from Example 1.2 and Problem 1.1). Thus to show that $n \longmapsto an + b$ is primitive recursive we need to show that h is primitive recursive. We can do this by writing

$$h(n) = \text{mult}(C_a(n), U_1^1(n)),$$

Notice how we need to use a projection function — namely the identity function — in order to obtain a formula that satisfies the definition of substitution even when there is only one variable.

demonstrating that h is obtained by substitution from the primitive recursive functions mult, C_a and U_1^1 (by Example 1.2, Problem 1.1 and Definition 1.1). Hence h is primitive recursive. Therefore

$$an + b = \text{add}(h(n), C_b(n))$$

and so $n \longmapsto an + b$ is primitive recursive since it is obtained by substitution from the primitive recursive functions add, h and C_b.

Notice that we needed to use the operation of substitution twice here, once to show that h is primitive recursive and then again to show that $n \longmapsto an + b$ is. Thus an alternative solution would be to write

$$an + b = \text{add}(\text{mult}(C_a(n), U_1^1(n)), C_b(n))$$

and to deduce that $n \longmapsto an + b$ is obtained using two levels of substitution from the primitive recursive functions add, mult, C_a, U_1^1 and C_b, so that $n \longmapsto an + b$ is primitive recursive.

Solution 1.3

We observe that

$$g_1(n_1, n_2) = f(C_a^2(n_1, n_2), U_1^2(n_1, n_2), U_2^2(n_1, n_2)),$$

so that g_1 is obtained by substitution from the primitive recursive functions f, C_a^2, U_1^2 and U_2^2. Therefore g_1 is primitive recursive.

Similarly by observing that

$$g_2(n_1, n_2) = f(U_1^2(n_1, n_2), C_a^2(n_1, n_2), U_2^2(n_1, n_2))$$
$$g_3(n_1, n_2) = f(U_1^2(n_1, n_2), U_2^2(n_1, n_2), C_a^2(n_1, n_2))$$

we can conclude that g_2 and g_3 are primitive recursive.

Solution 1.4

We observe that

$$g(n) = f(U_1^1(n), U_1^1(n)),$$

so that g is obtained by substitution from the primitive recursive functions f and U_1^1. Hence g is primitive recursive.

Solution 1.5

(a) Although with ordinary subtraction $(n - m) + 1$ will always equal $n - (m - 1)$, this result fails for cut-off subtraction. For instance, $(1 \dot{-} 0) + 1 = 1 + 1 = 2$, but $1 \dot{-} (0 \dot{-} 1) = 1 \dot{-} 0 = 1 \neq 2$.

(b) As the definition of $n \dot{-} m$ depends on whether $m \leq n$ or $n < m$, we need to compute $(n \dot{-} m) + m$ separately in each of these cases.

If $m \leq n$ then $n \dot{-} m = n - m$, so that $(n \dot{-} m) + m = n = \max(n, m)$.

If $n < m$ then $n \dot{-} m = 0$, so that $(n \dot{-} m) + m = m = \max(n, m)$.

It follows that

$$\max(n, m) = (n \dot{-} m) + m,$$

so that max is defined by substitution from the primitive recursive functions add and $\dot{-}$. Hence max is a primitive recursive function.

(c) As $n + m = \max(n, m) + \min(n, m)$, it follows that

$$\min(n, m) = (n + m) - \max(n, m)$$

and, as $\max(n, m) \leq n + m$, we have

$$\min(n, m) = (n + m) \dot{-} \max(n, m).$$

Thus min is defined by substitution from the primitive recursive functions $\dot{-}$, add and max. Hence min is a primitive recursive function.

There is often more than one way to show that a given function is primitive recursive. For example, here we could have used the fact that

$$\min(n, m) = n \dot{-} (n \dot{-} m).$$

Solution 1.6

We have

$$\mathrm{adf}(n, m) = (n \dot{-} m) + (m \dot{-} n).$$

Thus adf is obtained by substitution from the primitive recursive functions add and $\dot{-}$. Hence adf is a primitive recursive function.

Notice that

$$\mathrm{adf}(n, m) = \mathrm{add}(n \dot{-} m, g(n, m)),$$

where

$$g(n, m) = U_2^2(n, m) \dot{-} U_1^2(n, m),$$

so there are two levels of substitution needed here.

Solution 1.7

If $\chi_E(n) = 1$ then $\chi_E(n + 1) = 0$ and if $\chi_E(n) = 0$ then $\chi_E(n + 1) = 1$. Thus χ_E can be defined by the equations

$$\chi_E(0) = 1,$$

$$\chi_E(n + 1) = \overline{\mathrm{sg}}(\chi_E(n)).$$

Thus χ_E is primitive recursive since it is defined by primitive recursion from the constant 1 and the primitive recursive function $\overline{\mathrm{sg}}$. Hence E is a primitive recursive set.

Solution 1.8

Since A and B are primitive recursive sets, their characteristic functions χ_A and χ_B are primitive recursive.

Let n be a natural number.

Then $n \in A \cap B$ if $\chi_A(n) = 1$ and $\chi_B(n) = 1$ and $n \notin A \cap B$ if $\chi_A(n) = 0$ or $\chi_B(n) = 0$. Thus the characteristic function of $A \cap B$ is given by

$$\chi_{A\cap B}(n) = \chi_A(n) \times \chi_B(n).$$

Hence $\chi_{A\cap B}$ is defined by substitution from the primitive recursive functions mult, χ_A and χ_B, and so is a primitive recursive function. Thus $A \cap B$ is a primitive recursive set.

Also $n \in A \cup B$ if $\chi_A(n) + \chi_B(n) > 0$ and $n \notin A \cup B$ if $\chi_A(n) + \chi_B(n) = 0$. Thus the characteristic function of $A \cup B$ is given by

$$\chi_{A\cup B}(n) = \mathrm{sg}(\chi_A(n) + \chi_B(n)).$$

Hence $\chi_{A\cup B}$ is defined by substitution from the primitive recursive functions sg, add, χ_A and χ_B, and so is a primitive recursive function. Thus $A \cup B$ is a primitive recursive set.

Solution 1.9

Let (n, m) be an ordered pair of natural numbers.

We observe that if $n < m$ then $m \dot{-} n > 0$ while if $n \geq m$ then $m \dot{-} n = 0$. Thus the characteristic function of the relation $<$ is given by

$$\chi_<(n, m) = \mathrm{sg}(m \dot{-} n).$$

Thus $\chi_<$ is obtained by substitution from the primitive recursive functions sg and $\dot{-}$, and so is a primitive recursive function. Hence $<$ is a primitive recursive relation.

Next we observe that if $n \leq m$ then $n \dot{-} m = 0$ while if $n > m$ then $n \dot{-} m > 0$. Thus the characteristic function of the relation $\leq$ is given by

$$\chi_\leq(n, m) = \overline{\mathrm{sg}}(n \dot{-} m).$$

Thus $\chi_\leq$ is obtained by substitution from the primitive recursive functions $\overline{\mathrm{sg}}$ and $\dot{-}$, and so is a primitive recursive function. Hence $\leq$ is a primitive recursive relation.

Solution 1.10

By hypothesis, the characteristic functions χ_R and χ_S of the relations R and S are primitive recursive. The characteristic functions of the relations T, U and V are given by

Note the similarity of this solution to the proof of Theorem 1.2 and to Solution 1.8.

$$\chi_T(n_1, n_2, \ldots, n_k) = \overline{\mathrm{sg}}(\chi_R(n_1, n_2, \ldots, n_k)),$$
$$\chi_U(n_1, n_2, \ldots, n_k) = \chi_R(n_1, n_2, \ldots, n_k) \times \chi_S(n_1, n_2, \ldots, n_k),$$
$$\chi_V(n_1, n_2, \ldots, n_k) = \mathrm{sg}(\chi_R(n_1, n_2, \ldots, n_k) + \chi_S(n_1, n_2, \ldots, n_k)).$$

It follows that the relations T, U and V are primitive recursive, as their characteristic functions are obtained by substitution from the primitive recursive functions $\overline{\mathrm{sg}}$ and χ_R in the case of T, mult, χ_R and χ_S in the case of U and sg, add, χ_R and χ_S in the case of V.

Solution 1.11

Let S be the relation on ordered triples defined by

$$S(n_1, n_2, n_3) \iff n_1 \leq n_2.$$

Its characteristic function is given by

$$\chi_S(n_1, n_2, n_3) = \chi_{\leq}(n_1, n_2).$$

Since $\chi_{\leq}$ is primitive recursive (Problem 1.9), χ_S is a primitive recursive function (since it is obtained by substitution from the primitive recursive function $\chi_{\leq}$) and hence S is a primitive recursive relation.

Similarly the relation T on ordered triples defined by

$$T(n_1, n_2, n_3) \iff n_2 \leq n_3$$

is primitive recursive.

But

$$R(n_1, n_2, n_3) \iff S(n_1, n_2, n_3) \text{ and } T(n_1, n_2, n_3),$$

so R is a primitive recursive relation by Problem 1.10.

Solution 1.12

Since

$$h(n_1, n_2, n_3) = f(U_3^3(n_1, n_2, n_3), U_1^3(n_1, n_2, n_3), U_2^3(n_1, n_2, n_3))$$

we see that h is obtained by substitution from the primitive recursive function f and projection functions. Thus h is a primitive recursive function.

Solution 1.13

From *Unit 1*, we know that the factorial function $\text{fac} : n \longmapsto n!$ can be defined by primitive recursion as

$$\text{fac}(0) = 1,$$
$$\text{fac}(n+1) = \text{mult}(n+1, \text{fac}(n)).$$

Thus fac is defined by primitive recursion from the constant 1 and the primitive recursive functions mult and add using constants, and hence is primitive recursive.

We now consider the function $C : \mathbb{N}^2 \longmapsto \mathbb{N}$. For $r \leq n$,

$$C(n, r) = \frac{n!}{r!(n-r)!} = \text{quot}(n!,\ r! \times (n \dot{-} r)!).$$

Also, for $r > n$,

$$\text{quot}(n!,\ r! \times (n \dot{-} r)!) = \text{quot}(n!,\ r!) = 0$$

since $(n \dot{-} r)! = 0! = 1$.

Thus for all $n, r \in \mathbb{N}$ we can define the function C by

$$C(n, r) = \text{quot}(n!,\ r! \times (n \dot{-} r)!).$$

Hence C is defined by substitution from the primitive recursive functions quot, fac, mult and $\dot{-}$. Hence C is a primitive recursive function.

Solution 1.14

The formula for f is essentially the same as that in the proof of Theorem 1.4. We have just to replace the single variable n by the ordered l-tuple $(n_1, n_2, \ldots, n_l)$ and the sets $A_1, A_2, \ldots, A_k$ by the relations $R_1, R_2, \ldots, R_k$. Thus we have

$$\begin{aligned} f(n_1, n_2, \ldots, n_l) = {} & g_1(n_1, n_2, \ldots, n_l)\chi_{R_1}(n_1, n_2, \ldots, n_l) \\ & + g_2(n_1, n_2, \ldots, n_l)\chi_{R_2}(n_1, n_2, \ldots, n_l) \\ & + \cdots \\ & + g_k(n_1, n_2, \ldots, n_l)\chi_{R_k}(n_1, n_2, \ldots, n_l) \end{aligned}$$

and it follows that the function f is primitive recursive.

Solution 1.15

Let R_3 be the two-place relation defined by

$$R_3(n, m) \iff \text{not } R_1(n, m) \text{ and not } R_2(n, m).$$

Then

$$f(n, m) = \begin{cases} g_1(n, m), & \text{if } R_1(n, m), \\ g_2(n, m), & \text{if } R_2(n, m), \\ g_3(n, m), & \text{if } R_3(n, m). \end{cases}$$

It follows from Theorem 1.5 that f is primitive recursive provided that R_1, R_2 and R_3 are primitive recursive relations which are mutually exclusive and exhaustive.

From the definition of the relation R_3, at least one of the relations holds for all pairs (n, m), so that the three relations are exhaustive.

Furthermore, if $R_3(n, m)$ then neither $R_1(n, m)$ nor $R_2(n, m)$. Now suppose that $R_1(n, m)$ and $R_2(n, m)$. Then $\max(n, 2m) \geq 21$, so that either $n \geq 21$ in which case $3n + 4m \geq 63$, which is a contradiction of $R_2(n, m)$, or $m \geq 11$ in which case $3n + 4m \geq 44$, which is again a contradiction of $R_2(n, m)$. Therefore the three relations are mutually exclusive.

Thus the relations R_1, R_2 and R_3 are mutually exclusive and exhaustive.

Of course, if R_1, R_2 and R_3 hadn't been mutually exclusive and exhaustive then f wouldn't have been a function!

Next, the characteristic function of R_1 is

$$\chi_{R_1}(n, m) = \chi_{\leq}(21, \max(n, 2m)).$$

Thus χ_{R_1} is obtained by substitution from the primitive recursive functions $\chi_{\leq}$, max and mult using using constants. It follows that χ_{R_1} is a primitive recursive function. Thus R_1 is a primitive recursive relation.

Also the characteristic function of R_2 is

$$\chi_{R_2}(n, m) = \chi_{eq}(3n + 4m, 43).$$

Thus χ_{R_2} is obtained by substitution from the primitive recursive functions χ_{eq}, add and mult using using constants. It follows that χ_{R_2} is a primitive recursive function. Thus R_2 is a primitive recursive relation.

Finally R_3 is a primitive recursive relation by Problem 1.10.

Thus f is a primitive recursive function.

Solution 2.1

(a) $f(3, 2) = 2^3 3^2 = 8 \times 9 = 72$.

$f(2, 3) = 2^2 3^3 = 4 \times 27 = 108$.

(b) (i) $12 = 2^2 3^1$, so $(n, m) = (2, 1)$.

(ii) $16 = 2^4 3^0$, so $(n, m) = (4, 0)$.

(iii) $216 = 8 \times 27 = 2^3 3^3$, so $(n, m) = (3, 3)$.

Solution 2.2

(a) $f_3(3,2,1) = 2^3\,3^2\,5^1 = 8 \times 9 \times 5 = 360.$

$f_4(2,1,2,1) = 2^2\,3^1\,5^2\,7^1 = 4 \times 3 \times 25 \times 7 = 2100.$

(b) (i) $240 = 16 \times 3 \times 5 = 2^4\,3^1\,5^1$, so $(n_1, n_2, n_3) = (4,1,1)$.

(ii) $100 = 4 \times 25 = 2^2\,3^0\,5^2\,7^0$, so $(n_1, n_2, n_3, n_4) = (2,0,2,0)$.

Solution 2.3

Let $\mathbb{Z}^-$ be the set of negative integers: that is, $\mathbb{Z}^- = \{-1, -2, -3, \ldots\}$. Then $\mathbb{Z} = \mathbb{Z}^- \cup \mathbb{N}$. The function $f : \mathbb{Z}^- \longrightarrow \mathbb{N}$ given by $f(n) = |n|$ is one–one. Hence, since $\mathbb{Z}^-$ is infinite, it follows from Theorem 2.1 that $\mathbb{Z}^-$ is countably infinite. Therefore it follows from Theorem 2.3 that the set $\mathbb{Z}$ is countably infinite.

(An alternative method of proof is to observe that we can list the elements of $\mathbb{Z}$ in a single sequence as follows

$$\mathbb{Z} = \{0, -1, 1, -2, 2, -3, 3, \ldots\}$$

and we can use this listing to define, directly, a one–one correspondence between $\mathbb{Z}$ and $\mathbb{N}$.)

Solution 2.4

(a) $\beta(Z(5)) = 6 \times 5 - 3 = 27.$

(b) $\beta(S(4)) = 6 \times 4 = 24.$

(c) $\beta(C(5,1)) = 2^5\,3^1 + 1 = 32 \times 3 + 1 = 97.$

(d) $\beta(J(3,2,2)) = 2^3\,3^2\,5^2 + 2 = 8 \times 9 \times 25 + 2 = 1802.$

(e) $\beta(J(1,1,1)) = 2^1\,3^1\,5^1 + 2 = 2 \times 3 \times 5 + 2 = 32.$

Solution 2.5

The first step is to calculate the remainder when we divide the given number by 3. If the remainder is 0, the number codes either a Successor instruction or a Zero instruction according to whether it is also divisible by 6. If the remainder is 1 we look to see if we can write the number as $2^a\,3^b + 1$ with $a, b > 0$, and if the remainder is 2 we see whether we can write it in the form $2^a\,3^b\,5^c + 2$ with $a, b, c > 0$.

Note that once we know a number is divisible by 3, we need only check whether it is even to decide whether it is also divisible by 6.

(a) $24 = 6 \times 4$, so 24 codes the instruction $S(4)$.

(b) $25 = 24 + 1 = 2^3\,3^1 + 1$, so 25 codes the instruction $C(3,1)$.

(c) $26 = 24 + 2 = 2^3\,3^1 + 2$, so 26 does not have the correct form to be the code of a URM instruction.

(d) $27 = 6 \times 5 - 3$, so 27 codes the instruction $Z(5)$.

(e) $289 = 3 \times 96 + 1 = 2^5\,3^2 + 1$, so 289 codes the instruction $C(5,2)$.

(f) $272 = 3 \times 90 + 2 = 2^1\,3^3\,5^1 + 2$, so 272 codes the instruction $J(1,3,1)$.

(g) Since $50\,000 = 3 \times 16\,666 + 2$, if 50 000 codes a URM instruction then it must code a Jump instruction. But $3 \times 16\,666 = 49\,998$ which is clearly not divisible by 5 and so cannot have the form $2^a\,3^b\,5^c$ with $c > 0$. Hence 50 000 does not code a URM instruction.

Solution 2.6

The code numbers of the instructions are:

$$\beta(J(1,2,4)) = 2^1\,3^2\,5^4 + 2 = 2 \times 9 \times 625 + 2 = 11\,252,$$
$$\beta(S(2)) = 6 \times 2 = 12,$$
$$\beta(C(2,1)) = 2^2\,3^1 + 1 = 4 \times 3 + 1 = 13.$$

Hence $\gamma(P) = 2^{11\,252}\,3^{12}\,5^{13}$.

Solution 2.7

From Solution 2.6 we see that 11 252 codes the instruction $J(1,2,4)$. Since $6 = 6 \times 1$, we see that 6 codes the instruction $S(1)$. From Solution 2.4(e) we see that 32 codes the instruction $J(1,1,1)$. Hence the given number codes the URM program

1 $J(1,2,4)$
2 $S(1)$
3 $J(1,1,1)$

Solution 3.1

The function g satisfies the equations

$$g(n_1, n_2, \ldots, n_k, 0) = 1,$$
$$g(n_1, n_2, \ldots, n_k, z+1) = g(n_1, n_2, \ldots, n_k, z) \times f(n_1, n_2, \ldots, n_k, z+1).$$

This shows that g is defined by primitive recursion from the primitive recursive functions mult, f and add using constants. Hence it follows that g is primitive recursive.

Solution 3.2

We set out the values in the following table.

$g(n,z)$		z					
		1	2	3	4	5	6
n	4	2	2	2	2	2	2
	5	2	3	4	5	6	7
	6	2	3	4	5	6	7
	7	2	3	4	5	6	7
	8	2	3	4	5	6	7
	9	2	3	3	3	3	3

Solution 3.3

One way to prove the theorem is to mimic the proof of Theorem 3.4, as follows.

The function g can be defined as follows.

(a) $$g(n_1, n_2, \ldots, n_k, 0) = \begin{cases} 0, & \text{if } R(n_1, n_2, \ldots, n_k, 0), \\ 1, & \text{otherwise.} \end{cases}$$

(b) $$g(n_1, n_2, \ldots, n_k, z+1) = \begin{cases} g(n_1, n_2, \ldots, n_k, z), & \text{if } g(n_1, n_2, \ldots, n_k, z) \leq z, \\ z+1, & \text{if } g(n_1, n_2, \ldots, n_k, z) = z+1 \\ & \text{and } R(n_1, n_2, \ldots, n_k, z+1), \\ z+2, & \text{otherwise.} \end{cases}$$

The correctness of (a) should be clear. The cases in (a) are clearly mutually exclusive and exhaustive. Furthermore, using the result quoted after Problem 1.15, we can deduce that the relations defining the cases are primitive recursive. Also, since the formulas 0 and 1 for the functions on the right-hand side in (a) are constants, these functions are primitive recursive. Hence, by Theorem 1.5, the function in (a) is primitive recursive.

Let us now check that (b) is correct. Note first that if $g(n_1, n_2, \ldots, n_k, z) \leq z$ then the relation $R(n_1, n_2, \ldots, n_k, y)$ is satisfied for some $y \leq z$, and the smallest value of $y \leq z$ which satisfies this relation is also the smallest value of $y \leq z+1$ which satisfies this relation. So in this case the value of $g(n_1, n_2, \ldots, n_k, z+1)$ is the same as the value of $g(n_1, n_2, \ldots, n_k, z)$. Otherwise there is no value of $y \leq z$ that satisfies the relation and the value of $g(n_1, n_2, \ldots, n_k, z+1)$ is $z+1$ if $R(n_1, n_2, \ldots, n_k, z+1)$ holds and $z+2$ otherwise. Thus (b) is correct. Also, this discussion shows that the cases in (b) are mutually exclusive and exhaustive.

Furthermore, using the result quoted after Problem 1.15, we can deduce that the relations defining the cases in (b) are primitive recursive. Also, since the formulas $g(n_1, n_2, \ldots, n_k, z)$, $z+1$ and $z+2$ for the functions on the right-hand side are expressed in terms of $g(n_1, n_2, \ldots, n_k, z)$, the variable z and the primitive recursive function add using constants, these functions are primitive recursive. Hence, by Theorem 1.5, the function in (b) is primitive recursive.

Therefore g is defined by primitive recursion from functions known to be primitive recursive. It follows that g is a primitive recursive function.

Another way to prove the theorem is make use of Theorem 3.4 and the characteristic function χ_R of R.

From the definition of χ_R, we can express $\mu y \leq z\, R(n_1, n_2, \ldots, n_k, y)$ alternatively as

$$\mu y \leq z\, (\chi_R(n_1, n_2, \ldots, n_k, y) = 1).$$

To exploit Theorem 3.4 we need to turn $\chi_R(n_1, n_2, \ldots, n_k, y) = 1$ into something of the form $f(n_1, n_2, \ldots, n_k, y) = 0$. One way to do this is to use the signum-bar function, since

$$\chi_R(n_1, n_2, \ldots, n_k, y) = 1 \iff \overline{\mathrm{sg}}(\chi_R(n_1, n_2, \ldots, n_k, y)) = 0.$$

Now we define the function $f : \mathbb{N}^{k+1} \longrightarrow \mathbb{N}$ by

$$f(n_1, n_2, \ldots, n_{k+1}) = \overline{\mathrm{sg}}(\chi_R(n_1, n_2, \ldots, n_{k+1})).$$

This function is primitive recursive since it is defined by substitution from the primitive recursive functions $\overline{\mathrm{sg}}$ and χ_R.

So, finally we have, we have

$$g(n_1, n_2, \ldots, n_k, z) = \mu y \leq z\, (f(n_1, n_2, \ldots, n_k, y) = 0)$$

where f is primitive recursive. Hence, by Theorem 3.4, g is primitive recursive.

Solution 3.4

Consider the relation $R(m, y)$ given by

$$R(m, y) \iff m < 3^y.$$

Its characteristic function is

$$\chi_R(m, y) = \chi_<(m, \exp(3, y)),$$

which is obtained by substitution from the primitive recursive functions $\chi_<$ and exp using constants, and so is primitive recursive. Hence the relation R is primitive recursive.

Therefore, by Theorem 3.5, the function $g : \mathbb{N}^2 \longrightarrow \mathbb{N}$ given by

$$g(m, z) = \mu y \leq z\, R(m, y)$$

is primitive recursive.

We have $f(n) = \mu y\, R(n, y)$ and so to show that f is primitive recursive we need a suitable bound z on y expressed in terms of n (and perhaps earlier values of the function f). From the hint, $n < 3^n$ for all $n \in \mathbb{N}$, so that a suitable bound on y is simply n itself. Thus for all n we have

$$f(n) = g(n, n),$$

so that f is primitive recursive by Problem 1.4.

Solution 3.5

We shall show that f is defined by primitive recursion from known primitive recursive functions. It is the definition of $f(n+1)$ which requires attention, as $f(0)$ is simply defined using a constant.

Let $R(m, y)$ be the relation given by

$$R(m, y) \iff m < y \text{ and } y \text{ is not divisible by } 4.$$

The relation $<$ is primitive recursive (by Problem 1.9). We have

$$y \text{ is not divisible by } 4 \iff \operatorname{div}(y, 4) = 0.$$

One could also apply the 'not' part of Problem 1.10 to the relation 'y is divisible by 4'.

The characteristic function of the relation '$\operatorname{div}(y, 4) = 0$' is primitive recursive since it is obtained by substitution from the primitive recursive functions χ_{eq} and div using constants. Hence the relation 'y is not divisible by 4' is primitive recursive. Therefore, by Problem 1.10, the relation $R(m, y)$ is primitive recursive.

Hence, by Theorem 3.5, the function $g : \mathbb{N}^2 \longrightarrow \mathbb{N}$ given by

$$g(m, z) = \mu y \leq z\, R(m, y)$$

is primitive recursive.

We have $f(n+1) = \mu y\, R(f(n), y)$ and so to show that $f(n+1)$ is given by a primitive recursive function we require a bound z on y given by a primitive recursive function of n and $f(n)$. The biggest gap that can occur between $f(n)$ and $f(n+1)$ is 2. This happens when $f(n)+1$ is divisible by 4 and forces $f(n+1)$ to be $f(n)+2$; otherwise $f(n+1)$ is just $f(n)+1$. So a suitable bound z on y is $f(n)+2$. So letting $h : \mathbb{N} \longrightarrow \mathbb{N}$ be defined by $h(m) = m + 2$, so that h is primitive recursive (by Problem 1.2), we have

$$f(n+1) = \mu y \leq h(f(n))\, R(f(n), y) = g(f(n), h(f(n))).$$

Other bounds on y, such as $4n$, would do just as well. There are no prizes for a 'best' bound, even if there happens to be one!

Therefore we have

$$f(0) = 1,$$
$$f(n+1) = g(f(n), h(f(n))),$$

so that f is obtained by primitive recursion from the constant 1 and the primitive recursive functions g and h. Thus f is a primitive recursive function.

Solution 3.6

Since $45 = 3^2\, 5^1$, $\operatorname{len}(45) = 2$.

Since $700 = 2^2\, 5^2\, 7^1$, $\operatorname{len}(700) = 3$.

Since $2100 = 2^2\, 3^1\, 5^2\, 7^1$, $\operatorname{len}(2100) = 4$.

Solution 3.7

(a) Since $30 = 2^1\, 3^1\, 5^1$, $\chi_{Seq}(30) = 1$.

(b) Since $60 = 2^2\, 3^1\, 5^1$, $\chi_{Seq}(60) = 1$.

(c) Since $42 = 2^1\, 3^1\, 7^1$ (that is, it is divisible by 7 but not by 5), $\chi_{Seq}(42) = 0$.

(d) We do not need to factorize 200 200 completely to see that this number is divisible by 5 but not by 3. Hence $\chi_{Seq}(200\,200) = 0$.

Solution 3.8

As $360 = 2^3\,3^2\,5^1$, we see that $(360)_1 = 3$, $(360)_2 = 2$, $(360)_3 = 1$ and $(360)_j = 0$ otherwise.

As $243 = 3^5$, we see that $(243)_2 = 5$ and $(243)_j = 0$ otherwise.

As $462 = 2^1\,3^1\,7^1\,11^1$, we see that $(462)_1 = 1$, $(462)_2 = 1$, $(462)_4 = 1$, $(462)_5 = 1$ and $(462)_j = 0$ otherwise.

SOLUTIONS TO ADDITIONAL EXERCISES

Section 1

1 For $k \geq 1$, let C_a^k be the constant function of k variables with value a. We already know (Problem 1.1) that C_a^1 is a primitive recursive function. Now

$$C_a^k(n_1, n_2, \ldots, n_k) = C_a^1(n_1) = C_a^1(U_1^k(n_1, n_2, \ldots, n_k)).$$

Thus C_a^k is obtained by substitution from the primitive recursive function C_a^1 and the projection function U_1^k. Hence C_a^k is a primitive recursive function.

2 Put $A_k = \{n \in \mathbb{N} : n \neq k\}$. Its characteristic function χ_{A_k} is given by

$$\chi_{A_k}(n) = \overline{\text{sg}}(\chi_{eq}(n, k))$$

where χ_{eq} is the characteristic function of the primitive recursive relation of equality. Thus χ_{A_k} is obtained by substitution from the primitive recursive functions $\overline{\text{sg}}$ and χ_{eq} using constants. Therefore χ_{A_k} is a primitive recursive function and hence A_k is a primitive recursive set.

Put $B_k = \{n \in \mathbb{N} : n > k\}$. Its characteristic function χ_{B_k} is given by

$$\chi_{B_k}(n) = \prod_{r=0}^{k} \chi_{A_r}(n).$$

The symbol $\prod$ indicates a product, so that

$$\prod_{r=0}^{k} \chi_{A_r}(n) = \chi_{A_0}(n) \times \chi_{A_1}(n) \times \cdots \times \chi_{A_k}(n).$$

Thus χ_{B_k} is obtained by substitution from the primitive recursive functions mult and $\chi_{A_0}, \chi_{A_1}, \ldots, \chi_{A_k}$. Therefore χ_{B_k} is a primitive recursive function and hence B_k is a primitive recursive set. (Alternatively,

$$\chi_{B_k}(n) = \chi_{<}(k, n)$$

where $\chi_{<}$ is the characteristic function of the primitive recursive relation $<$. Therefore χ_{B_k} is primitive recursive since it is obtained by substitution from the primitive recursive function $\chi_{<}$ using constants.)

3 We have

$$A \Delta B = (A \cap (\mathbb{N} \setminus B)) \cup (B \cap (\mathbb{N} \setminus A)).$$

Hence if A and B are primitive recursive sets, it follows from Theorem 1.2 and Problem 1.8 that the set $A \Delta B$ is also primitive recursive.

4 The definition is of the form

$$f(n_1,n_2,n_3)=\begin{cases} g_1(n_1,n_2,n_3), & \text{if } R_1(n_1,n_2,n_3),\\ g_2(n_1,n_2,n_3), & \text{if } R_2(n_1,n_2,n_3),\\ g_3(n_1,n_2,n_3), & \text{if } R_3(n_1,n_2,n_3),\end{cases}$$

where

$$\begin{aligned} R_1(n_1,n_2,n_3) &\iff n_1+12\le\min(n_2,n_3),\\ R_2(n_1,n_2,n_3) &\iff 2n_1+3n_2+4n_3=80,\\ R_3(n_1,n_2,n_3) &\iff \text{not } R_1(n_1,n_2,n_3) \text{ and not } R_2(n_1,n_2,n_3).\end{aligned}$$

It follows from Theorem 1.5 that f is primitive recursive if R_1, R_2 and R_3 are primitive recursive relations which are mutually exclusive and exhaustive and g_1, g_2 and g_3 are primitive recursive functions.

From the definition of the relation R_3, at least one of the relations holds for all $(n_1,n_2,n_3)\in\mathbb{N}^3$, so that the three relations are exhaustive.

Furthermore, if $R_3(n_1,n_2,n_3)$ then neither $R_1(n_1,n_2,n_3)$ nor $R_2(n_1,n_2,n_3)$. Now suppose $R_1(n_1,n_2,n_3)$ and $R_2(n_1,n_2,n_3)$. Then $n_2\ge 12$ and $n_3\ge 12$ so that $2n_1+3n_2+4n_3\ge 36+48>80$ which is a contradiction of $R_2(n_1,n_2,n_3)$. Therefore the three relations are mutually exclusive.

Thus the relations R_1, R_2 and R_3 are mutually exclusive and exhaustive.

Next, the characteristic function of R_1 is given by

$$\chi_{R_1}(n_1,n_2,n_3)=\chi_{\le}(n_1+12,\min(n_2,n_3)).$$

This is primitive recursive as it is obtained by substitution from the primitive recursive functions $\chi_{\le}$, add and min using constants.

Also the characteristic function of R_2 is given by

$$\chi_{R_2}(n_1,n_2,n_3)=\chi_{eq}(2n_1+3n_2+4n_3,80),$$

This is primitive recursive as it is obtained by substitution from the primitive recursive functions χ_{eq}, add and mult using constants.

Thus the relations R_1 and R_2 are primitive recursive.

It follows from Problem 1.10 that the relation R_3 is primitive recursive.

Finally,

$$\begin{aligned} g_1(n_1,n_2,n_3)&=2n_2,\\ g_2(n_1,n_2,n_3)&=\exp(n_1,n_3),\\ g_3(n_1,n_2,n_3)&=5,\end{aligned}$$

and these are primitive recursive functions.

Thus f is a primitive recursive function.

g_1 is primitive recursive since it is obtained from mult using constants; g_2 is primitive recursive since exp is; g_3 is primitive recursive since it is a constant function.

Section 2

1 (a) $f_3(4,2,4)=2^4\,3^2\,5^4=16\times 9\times 625=90\,000.$
$f_3(4,3,2)=2^4\,3^3\,5^2=16\times 27\times 25=10\,800.$

(b) (i) As $100=2^2\,3^0\,5^2$, we see that $f_3(2,0,2)=100$.

(ii) As $480=32\times 3\times 5=2^5\,3^1\,5^1$, we see that $f_3(5,1,1)=480$.

(iii) As 490 is divisible by 7, it is not of the form $2^{n_1}\,3^{n_2}\,5^{n_3}$ and so is not a value of f_3.

2 (a) Suppose that, for each $k \in \mathbb{N}$, the set A_k is countably infinite. Then for each $k \in \mathbb{N}$ there is a one–one function $f_k : A_k \longrightarrow \mathbb{N}$. We show now how to define a one–one function

$$f : \bigcup_{k \in \mathbb{N}} A_k \longrightarrow \mathbb{N}.$$

Let $x \in \bigcup_{k \in \mathbb{N}} A_k$ and let k be the smallest natural number such that $x \in A_k$. Then $f_k(x)$ is defined and is a natural number. We put

$$f(x) = 2^k \, 3^{f_k(x)}.$$

We show now that f is a one–one function. Suppose that $x, y \in \bigcup_{k \in \mathbb{N}} A_k$ and $f(x) = f(y)$. Let k and l be the smallest natural numbers such that $x \in A_k$ and $y \in A_l$, so that $f(x) = 2^k 3^{f_k(x)}$ and $f(y) = 2^l 3^{f_l(y)}$. Then, since $f(x) = f(y)$ we have $2^k \, 3^{f_k(x)} = 2^l \, 3^{f_l(y)}$, so that, by the Fundamental Theorem of Arithmetic, $k = l$ and $f_k(x) = f_l(y)$. Thus $x, y \in A_k$ and $f_k(x) = f_k(y)$. Since f_k is a one–one function, it follows that $x = y$. Hence the function f is one–one and it follows from Theorem 2.1 that the infinite set $\bigcup_{k \in \mathbb{N}} A_k$ is countably infinite.

The notations $\bigcup_{k \in \mathbb{N}}$ and $\bigcup_{k \in \mathbb{N}}$ are equivalent.

(b) For each $k \geq 2$, put

$$A_k = \{x \in \mathbb{R} : x > 0 \text{ and } x^k \in \mathbb{N}\}.$$

Thus $A_k = \{\sqrt[k]{1}, \sqrt[k]{2}, \sqrt[k]{3}, \ldots\}$. The function $f_k : A_k \longrightarrow \mathbb{N}$ given by $f_k(x) = x^k$ is one–one, so that, for each $k \geq 2$, the infinite set A_k is countably infinite (by Theorem 2.1). To use the result of part (a), we need to assign countably infinite sets to A_0 and A_1, which we can do without upsetting things by putting $A_0 = A_1 = A_2$. We then have that the set of root numbers is $\bigcup_{k \in \mathbb{N}} A_k$ and this set is countably infinite by the result of part (a).

Note that, for each $n > 0$, $x^k = n$ has a unique positive solution x, which we write as $\sqrt[k]{n}$.

3 (a) Using the hint that each positive rational number x can be written in a unique way as a_x/b_x where a_x and b_x are relatively prime positive integers, we can define a function $f : \mathbb{Q}^+ \longrightarrow \mathbb{N}$ by

$$f(x) = 2^{a_x} 3^{b_x},$$

which is clearly one–one. By Theorem 2.1, the infinite set $\mathbb{Q}^+$ is countably infinite.

(b) We can show that the infinite set $\mathbb{Q}^-$ of negative rationals is countably infinite in a similar way to part (a). Each negative rational number y can be written in a unique way as $-a_y/b_y$ where a_y and b_y are relatively prime positive integers. The function $g : \mathbb{Q}^- \longrightarrow \mathbb{N}$ given by

$$g(y) = 2^{a_y} 3^{b_y},$$

is clearly one–one and so, by Theorem 2.1, $\mathbb{Q}^-$ is countably infinite.

We can extend the definition of g to cope with the rational number 0 by setting $g(0) = 0$, so that g is then a one–one function from the set $\mathbb{Q}_0^-$ of all non-positive rational numbers to $\mathbb{N}$, which then shows that the set $\mathbb{Q}_0^-$ is countably infinite.

As $\mathbb{Q} = \mathbb{Q}^+ \cup \mathbb{Q}_0^-$, Theorem 2.3 gives that $\mathbb{Q}$ is countably infinite.

4 It is clear that the set $\mathscr{P}(\mathbb{N})$ of all subsets of $\mathbb{N}$ is infinite as it includes for each $k \in \mathbb{N}$ the set $\{k\}$ whose only member is k. As suggested by the hint, we shall use a diagonal argument to show that there is no function from $\mathbb{N}$ to $\mathscr{P}(\mathbb{N})$ which is onto.

Suppose that $\Phi : \mathbb{N} \longrightarrow \mathscr{P}(\mathbb{N})$. Define a subset A of $\mathbb{N}$ by the rule

$$n \in A \iff n \notin \Phi(n),$$

that is, n is in A if and only if n is not in the subset $\Phi(n)$. Then A cannot be one of the values taken by the function Φ, for if it were, A would be the subset $\Phi(n)$ for some $n \in \mathbb{N}$, and for this value of n,

$$\begin{aligned} n \in A \quad &\text{if and only if} \quad n \notin \Phi(n) \quad \text{(by the definition of } A\text{)} \\ &\text{if and only if} \quad n \notin A \quad \text{(as } \Phi(n) = A\text{)}, \end{aligned}$$

which is a contradiction.

Thus there is no function $\Phi : \mathbb{N} \longrightarrow \mathscr{P}(\mathbb{N})$ which is onto, from which it follows that the set $\mathscr{P}(\mathbb{N})$ is not countably infinite.

If you cannot quite see why we described the above as a diagonal argument, perhaps the following picture will help.

	0	1	2	3	4	5	
$\Phi(0)$	$\underline{I}$	I	O	I	O	O	$\cdots$
$\Phi(1)$	I	$\underline{I}$	I	O	I	O	$\cdots$
$\Phi(2)$	O	O	$\underline{O}$	O	I	I	$\cdots$
$\Phi(3)$	I	O	I	$\underline{O}$	I	O	$\cdots$
$\Phi(4)$	O	O	O	O	$\underline{O}$	O	$\cdots$
$\Phi(5)$	I	O	O	O	I	$\underline{I}$	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	

If n belongs to the subset $\Phi(m)$, place an I (for 'in') in the appropriate entry of the table and otherwise put an O (for 'out'). Then the subset A is defined so that it differs from each $\Phi(m)$ along the diagonal.

5 (a) (i) $\beta(Z(7)) = 6 \times 7 - 3 = 39$.

(ii) $\beta(S(8)) = 6 \times 8 = 48$.

(iii) $\beta(C(5,2)) = 2^5\,3^2 + 1 = 289$.

(iv) $\beta(J(1,5,3)) = 2^1\,3^5\,5^3 + 2 = 60\,752$.

(b) (i) As $72 = 6 \times 12$, we see that 72 codes the instruction $S(12)$.

(ii) As $73 = 72 + 1 = 8 \times 9 + 1 = 2^3\,3^2 + 1$, we see that 73 codes the instruction $C(3,2)$.

(iii) As $74 = 72 + 2$ and 72 is divisible by 2 and 3 but not by 5, we see that 74 does not code a URM instruction.

(iv) As $75 = 6 \times 13 - 3$, we see that 75 codes the instruction $Z(13)$.

(v) As $152 = 150 + 2 = 2^1\,3^1\,5^2 + 2$, we see that 152 codes the instruction $J(1,1,2)$.

6 We note that $\beta(Z(1)) = 3$ and $\beta(Z(n)) > 3$ for $n > 1$. Also $\beta(S(n)) \geq 6$, $\beta(C(m,n)) \geq 7$ and $\beta(J(m,n,q)) \geq 32$ for all $m, n, q \geq 1$. Thus 3 is the smallest number which codes a URM instruction.

Section 3

1 We must show that the characteristic function of the set is primitive recursive. This function is given by

$$n \longmapsto \overline{\mathrm{sg}}(\mathrm{div}(n, 3)),$$

and so is obtained by substitution from the primitive recursive functions $\overline{\mathrm{sg}}$ and div using constants. Hence it is primitive recursive.

2 We have $\sigma(0) = 0$ and, if $n > 0$,

$$\sigma(n) = \sum_{k=1}^{n} (k \times \mathrm{div}(n, k)).$$

The function $(n, k) \longmapsto k \times \mathrm{div}(n, k)$ is primitive recursive since it is obtained by substitution from the primitive recursive functions mult and div. Hence, by Theorem 3.1, the function $g : \mathbb{N}^2 \longrightarrow \mathbb{N}$ given by

$$g(n, z) = \begin{cases} 0, & \text{if } z = 0, \\ \displaystyle\sum_{k=1}^{z} (k \times \mathrm{div}(n, k)), & \text{otherwise,} \end{cases}$$

is primitive recursive. As $\sigma(n) = g(n, n)$, we see from Problem 1.4 that σ is a primitive recursive function.

3 (a) $g(64, 2) = \mu y \leq 2\,(y^3 = 64) = 3$, as there is no y with $y \leq 2$ and $y^3 = 64$.

(b) $g(125, 10) = \mu y \leq 10\,(y^3 = 125) = 5$.

(c) $g(25, 10) = \mu y \leq 10\,(y^3 = 25) = 11$, as there is no y with $y \leq 10$ and $y^3 = 25$ (and, indeed, no $y \in \mathbb{N}$ such that $y^3 = 25$).

4 Let $R(m, y)$ be the relation given by

$$R(m, y) \iff m \leq y \text{ and } y \text{ is a multiple of } 5.$$

The relation $\leq$ is primitive recursive by Problem 1.9. Also

$$y \text{ is a multiple of } 5 \iff \mathrm{div}(y, 5) = 1.$$

The characteristic function of the relation '$\mathrm{div}(y, 5) = 1$' is primitive recursive since it is obtained by substitution from the primitive recursive functions χ_{eq} and div using constants. Hence the relation 'y is a multiple of 5' is primitive recursive. Therefore, by Problem 1.10, the relation $R(m, y)$ is primitive recursive and, by Theorem 3.5, the function $g : \mathbb{N}^2 \longrightarrow \mathbb{N}$ given by

$$g(m, z) = \mu y \leq z\, R(m, y)$$

is primitive recursive. A suitable bound z for y in terms of n by which we can be sure of finding $f(n)$ is $n + 6$. So letting $h : \mathbb{N} \longrightarrow \mathbb{N}$ be given by $h(n) = n + 6$, which is primitive recursive (by Problem 1.2), we have

$$f(n) = g(n, h(n)).$$

As f is obtained by substitution from the primitive recursive functions g and h, f is primitive recursive.

There are other ways of showing f to be primitive recursive, for instance a definition by cases depending on the remainder when n is divided by 5.

5 (a) (i) As $60 = 2^2\,3^1\,5^1$, it follows that $\text{len}(60) = 3$.

(ii) The prime divisors of 20! are all the prime numbers less than 20, that is, 2, 3, 5, 7, 11, 13, 17 and 19. Thus $\text{len}(20!) = 8$.

(b) (i) As $420 = 2^2\,3^1\,5^1\,7^1$, we see that 420 codes the sequence $2, 1, 1, 1$. Therefore $\chi_{Seq}(420) = 1$.

(ii) It can be seen that 4900 is divisible by 5 but not by 3 and hence it cannot code a sequence. Therefore $\chi_{Seq}(4900) = 0$.

(iii) We noted in part (a) that 20! has prime divisors 2, 3, 5, 7, 11, 13, 17 and 19. This is a sequence of prime numbers starting at 2 with no gaps. Hence 20! codes a sequence. Therefore $\chi_{Seq}(20!) = 1$.

(c) (i) As $720 = 2^4\,3^2\,5^1$, the exponent of 3 is 2 so that $(720)_2 = 2$.

(ii) As $7007 = 7^2 \times 143$ and 7 does not divide 143 we see (noting that $p_4 = 7$) that $(7007)_4 = 2$.

(iii) Fortunately there is no need to factorize 3275 346 933 to see that it is not divisible by 2 and hence that $(3275\,346\,933)_1 = 0$.

6 (a) We have

$$G(0) = 2^{F(0)}3^{F(1)} = 2^0\,3^1 = 3.$$

Also

$$G(n+1) = 2^{F(n+1)}3^{F(n+2)} = 2^{F(n+1)}3^{F(n+1)+F(n)}$$

using the definition of the Fibonacci sequence. Now

$$F(n) = (G(n))_1 \quad \text{and} \quad F(n+1) = (G(n))_2.$$

Thus we obtain the following recursion equation for G.

$$G(n+1) = 2^{(G(n))_2}3^{(G(n))_2+(G(n))_1}.$$

Thus we have

$$G(0) = 3,$$

$$G(n+1) = 2^{(G(n))_2}3^{(G(n))_2+(G(n))_1},$$

so that G is defined by primitive recursion from the constant 3 and the primitive recursive functions mult, exp, $(a, b) \longmapsto (a)_b$ and add using constants. Hence G is primitive recursive.

(b) The function F is given by

$$F(n) = (G(n))_1.$$

Thus F is obtained by substitution from the primitive recursive functions $(a, b) \longmapsto (a)_b$ and G using constants. It follows that F is primitive recursive.

7 Let g be the function defined for $n \geq 2$ by

$$g(n) = \text{the largest value of } y \text{ such that } p(y) \text{ is a divisor of } n.$$

Recall that $p(y) = p_y$ is the yth prime number.

Then, for $n \geq 2$, $f(n) = p(g(n))$. We know that p is primitive recursive (Theorem 3.6), so to show that f is primitive recursive we are going to need to show that g is primitive recursive.

It is tempting to say that

$$g(n) = \mu y \leq n\,(\mathrm{div}(n, p(y)) = 1 \text{ and } \mathrm{div}(n, p(y+1)) = 0)$$

as it would then follow that g is primitive recursive. But unfortunately this definition of g is not correct. For example, as 10 is divisible by 2 but not by 3 (that is, $\mathrm{div}(10, p(1)) = 1$ and $\mathrm{div}(10, p(2)) = 0$) this formula gives $g(10) = 1$, but the largest prime number which is a divisor of 10 is 5, that is, $p(3)$ not $p(1)$.

One way round this difficulty is to define g by searching for the largest prime which divides n downwards rather than upwards. We know that if $p(y)$ is a divisor of n then $y < n$. So the largest prime which divides n is $p(n \dot{-} k)$ where k is the smallest positive integer such that $p(n \dot{-} k)$ is a divisor of n. Hence if we define the function h by

$$h(n) = \mu k \leq n\,(\mathrm{div}(n, p(n \dot{-} k)) = 1)$$

then the function g is given by

$$g(n) = n \dot{-} h(n).$$

The relation '$\mathrm{div}(n, p(n \dot{-} k)) = 1$' is primitive recursive since its characteristic function is obtained by substitution from the primitive recursive functions χ_{eq}, div, p and $\dot{-}$ using constants. Hence h is primitive recursive by Theorem 3.5. Therefore g is also primitive recursive since it is obtained by substitution from the primitive recursive functions $\dot{-}$ and h.

Thus we can define f by

$$f(n) = \begin{cases} p(g(n)), & \text{if } n \geq 2, \\ n, & \text{otherwise.} \end{cases}$$

The function $n \longmapsto p(g(n))$ is primitive recursive since it is obtained by substitution from the primitive recursive functions p and g. The identity function $n \longmapsto n$ is primitive recursive. Since the relation $\geq$ is primitive recursive, using the remark following Problem 1.15 we can deduce that the relations defining the cases are primitive recursive. Moreover, these relations are clearly mutually exclusive and exhaustive. So, by Theorem 1.5, f is primitive recursive.

INDEX

$\dot{-}$ 10
$\iff$ 16
$\longleftrightarrow$ 22
$\prod$ 34
$f|_Y$ 25
f^{-1} 22
nRm 14
$\binom{n}{r}$ 19
$p(n)$ 33
p_k 25
$[r]$ 18
$(n)_j$ 42
$A \Delta B$ 48
C_a 7
C_a^k 7
$R(n,m)$ 14
$R(n_1, n_2, \ldots, n_k)$ 15
I 27
P 28
$\mathbb{C}$ 23
$\mathbb{N} \setminus A$ 13
$\mathbb{Q}$ 23, 49
$\mathbb{Q}^+$ 49
$\mathbb{Q}^-$ 62
$\mathbb{Q}_0^-$ 62
$\mathbb{R}$ 23
$\mathbb{R}^2$ 23
$\mathbb{Z}^+$ 12
$\mathbb{Z}^-$ 56
$\mathscr{P}(\mathbb{N})$ 49
β 27
$\beta(I)$ 27
γ 29
$\gamma(P)$ 29
μy 35
$\mu y \leq z\,(f(n_1, n_2, \ldots, n_k, y) = 0)$ 36
$\mu y \leq z\, R$ 35, 38
$\sigma(n)$ 49
χ_A 12
χ_R 15

add 7
adf 11
div 33
exp 10
len 41
max 11
sg 12
$\overline{\text{sg}}$ 12
min 11
mult 8
pred 10
quot 17
rem 17
eq 15
Cinstr 43
Instr 43
Jinstr 43
Pr 33
Prog 45
Seq 40
Sinstr 43
Zinstr 43

absolute difference 11

basic primitive recursive functions 6
bijection 22
binomial coefficient 19
bounded minimization
    on a function 36, 37
    on a relation 38
bounded product 34
bounded search 36
bounded summation 34

Cantor 22
Cantor's diagonal argument 31
characteristic function
    of a relation 15, 16
    of a set 12
code
    of a sequence of positive integers 40
    of a URM instruction 27
    of a URM program 29
complement 13
constant function 7
constants
    function defined by using 9
    relation defined by using 16
correspondence
    one–one 22
countable 23
countably infinite 23
cut-off subtraction 10

definition by cases 11, 20
denumerable 23
diagonal argument 31

equality of functions 32
exponentiation 10

Fibonacci sequence 50
floor 18
function
    characteristic 12, 15, 16
    defined by cases 11, 20
    defined using constants 9
    inverse 22
    one–one 22
    onto 22
    primitive recursive 6
    restriction 25

integer part 18
inverse function 22

k-place relation 15

length 41

minimization 36
    bounded 36–38
mutually exclusive and exhaustive 21

one–one correspondence 22
one–one function 22
onto function 22

partition 20
permutation 17
predecessor 10
primitive recursion 5
primitive recursive
    function 6
    function defined by cases 20
    relation 16
    set 12

quotient function 17

relation 14–16
    defined using constants 16
    primitive recursive 15, 16
    URM-computable 15, 16
remainder function 17
restriction function 25
root number 48

search 36
set
    countably infinite 23
    primitive recursive 12
    URM-computable 12
signum 12
signum-bar 12
substitution 4
symmetric difference 48

URM-computable relation 15, 16
URM-computable set 12